GNNs for HL-LHC Tracking

ExaTrkX @ Berkeley Lab

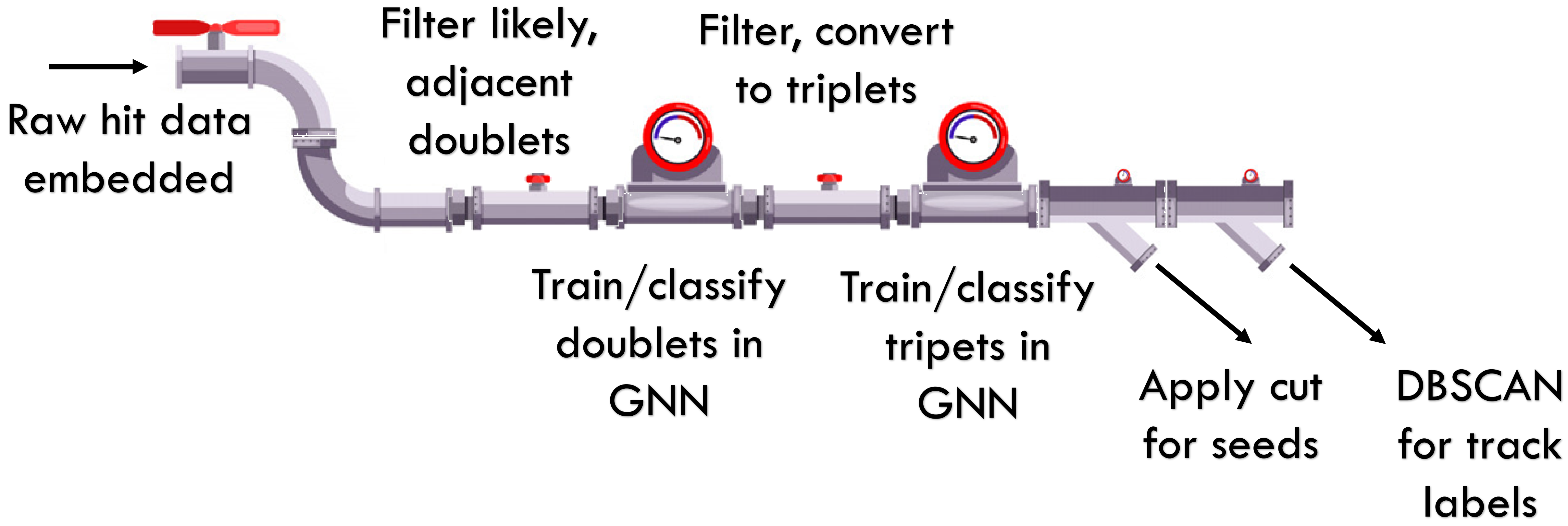
Daniel Murnane

Goal

Sub-second processing of HL-LHC hit data into:

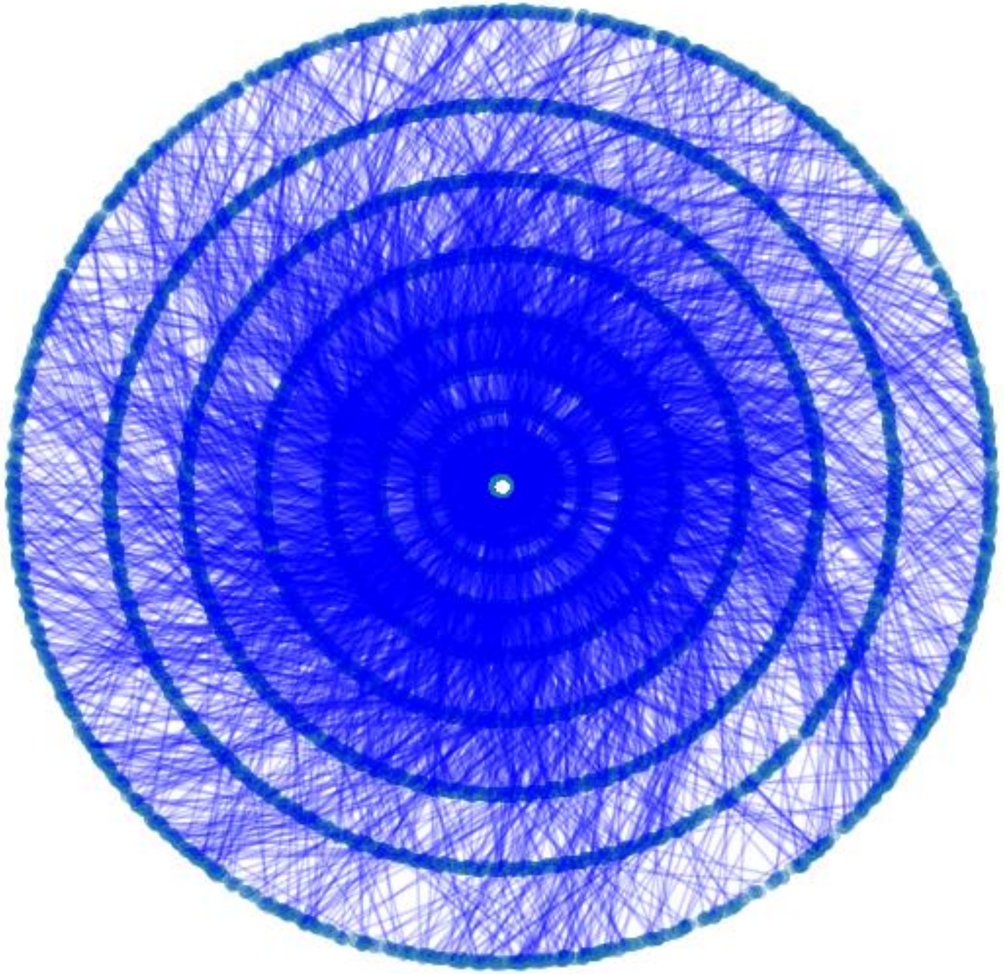
- Seeds (i.e. triplets) for further processing with traditional techniques, AND/OR
- Tracks, where each hit is assigned to exactly one track

The Current Pipeline



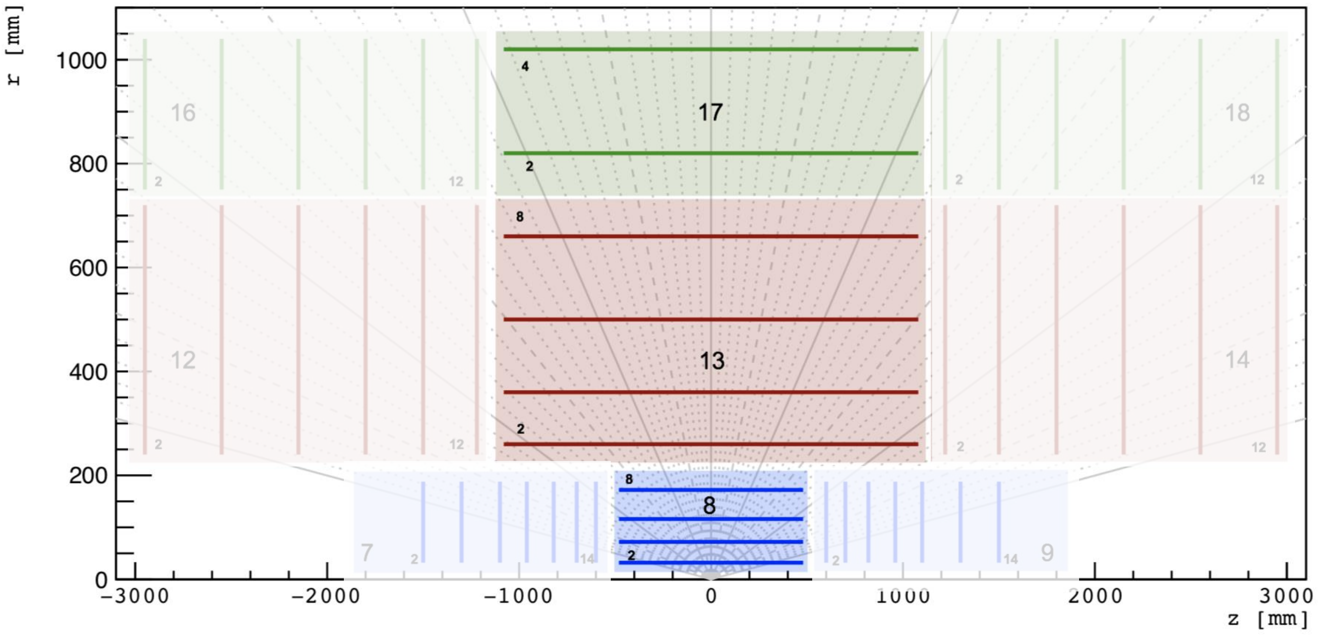
Dataset

- "TrackML Kaggle Competition" dataset
- Generated by simulation
- 8000 collisions to train on
- Each collision has up to 100,000 hits of around 10,000 particles



Dataset

- Ideal final result is a "TrackML score"
  $$S \in [0,1]$$
- All hits belonging to same track labelled with same unique label $\Rightarrow S = 1$
- We use the barrel as a test case, and ignore noise



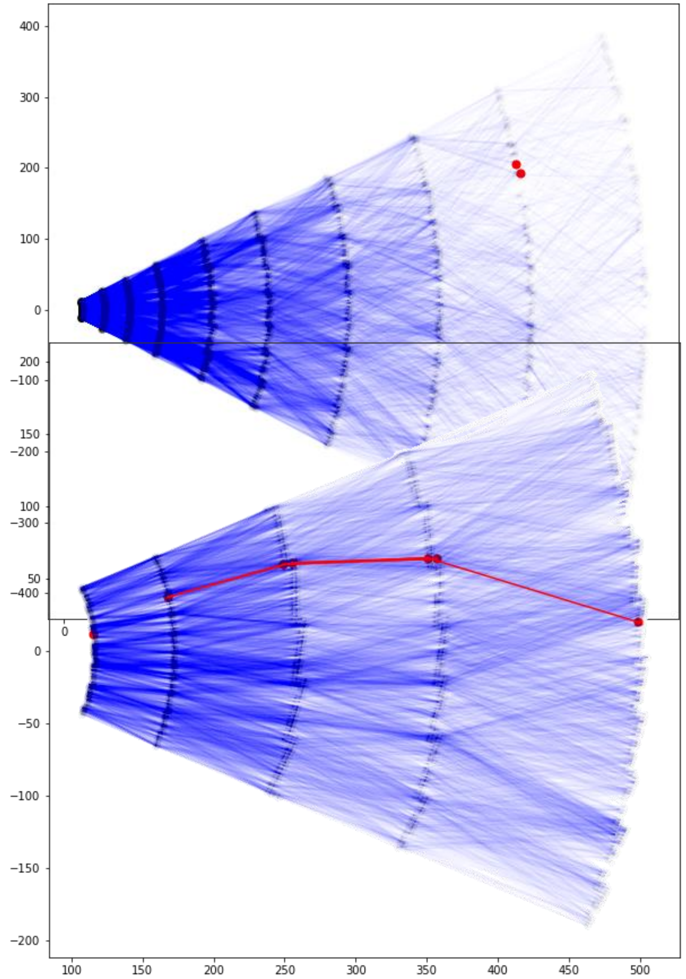
Embedding + MLP Construction

- Won't give any detail (Nick's talk next on embeddings)
- Generally:

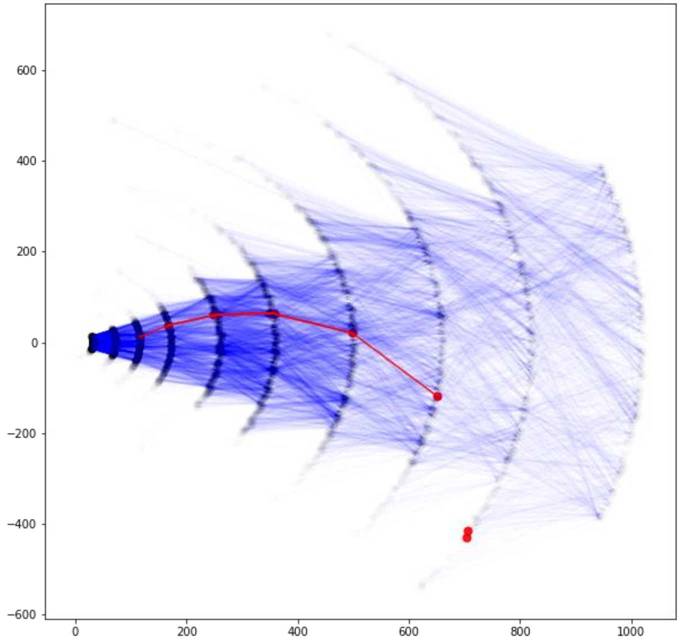
1. For each hit in event, embed features (co-ordinates, cell direction data, etc.) into N-dimensional space
2. Associate hits from same tracks as close in N-dimensional distance
3. Score each hit within embedding neighbourhood against the "seed" hit at centre
4. Filter by score, to create a set of doublets for the neighbourhood
5. All doublets in event generate a graph,
   converted to a directed graph (by ordering layers)

Segmentation

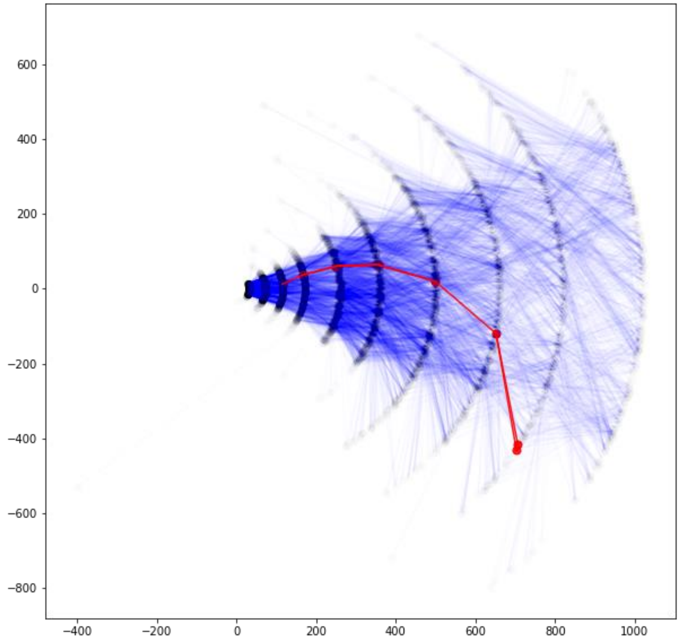
A full graph from the embedding does not fit on a single GPU. Therefore the event graphs are segmented, according to how large the GNN model is expected to be.



Hard cut



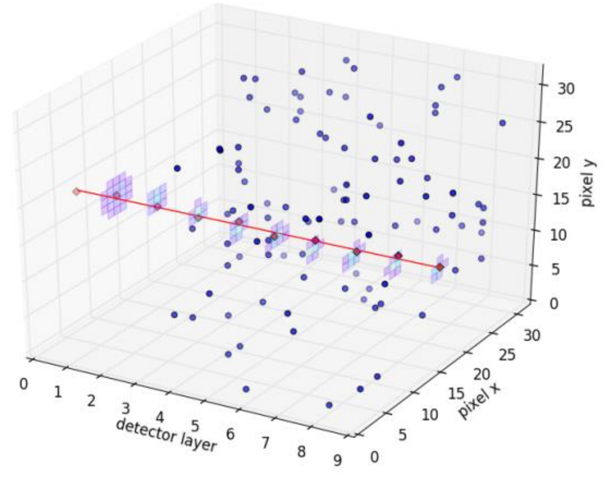
One-directional soft cut

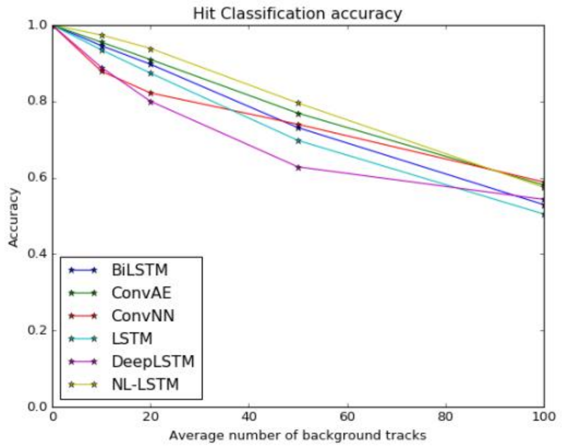


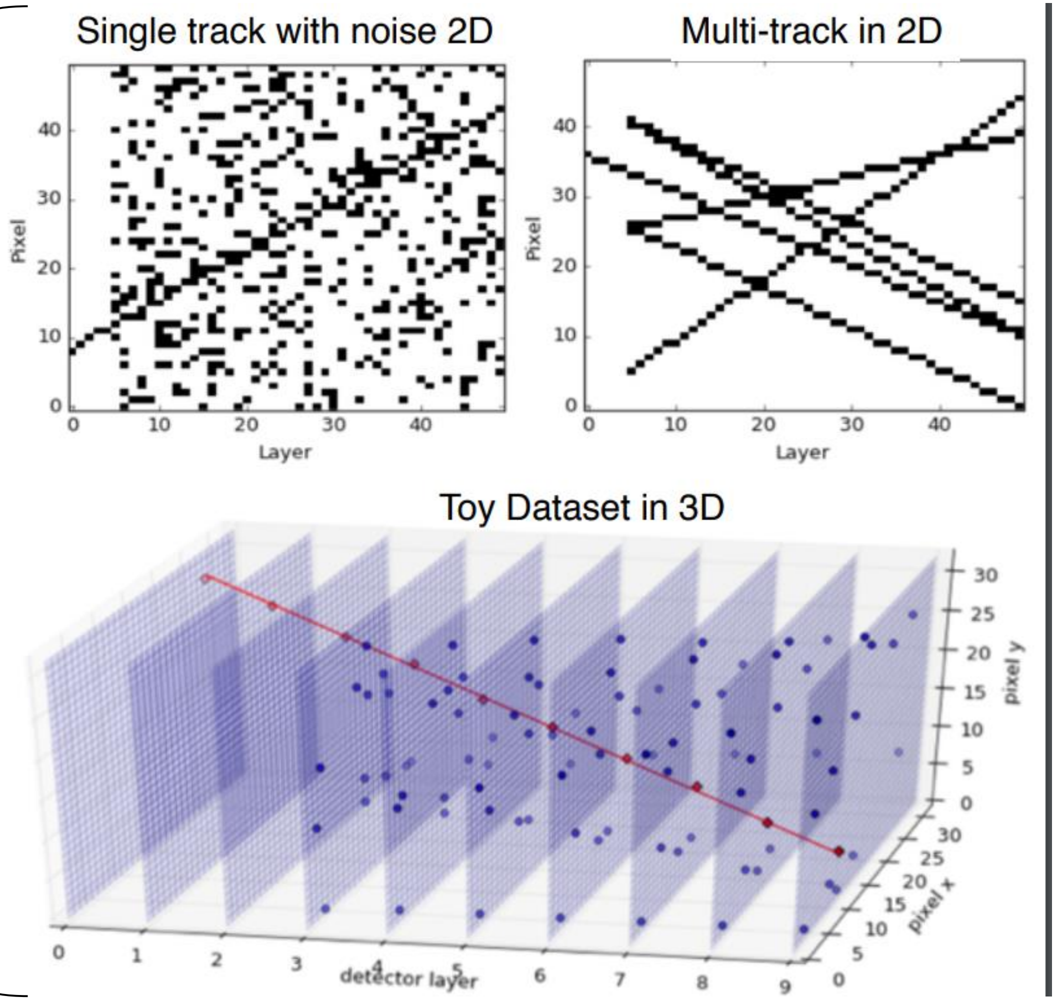
Bi-directional soft cut

Previous ML Approaches

- Tracks as **images (CNN)**
- Tracks as **sequences of points (LSTM)**





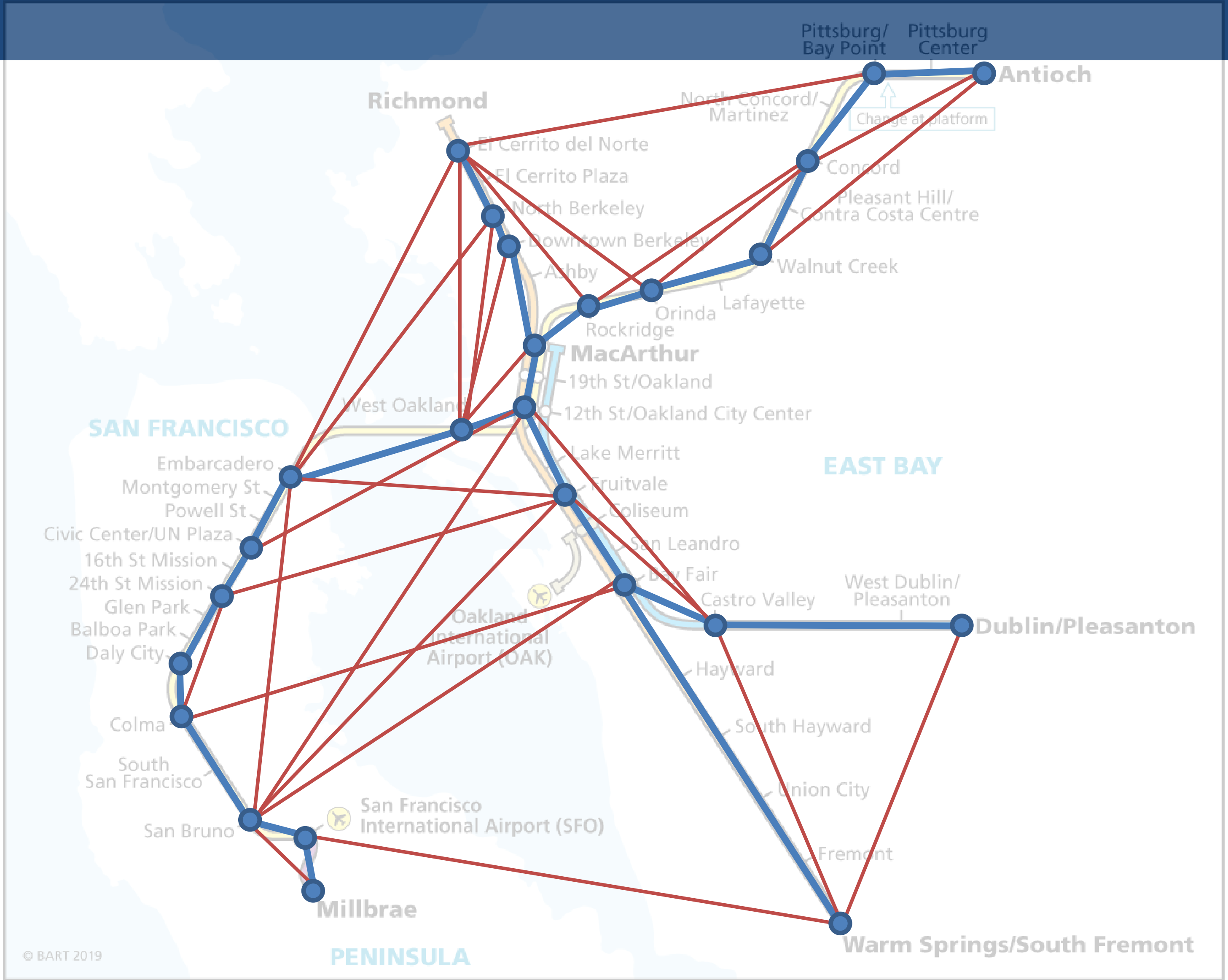


Graph Neural Network for Edge Classification

Classify edges with score between [0,1]

score > cut: true

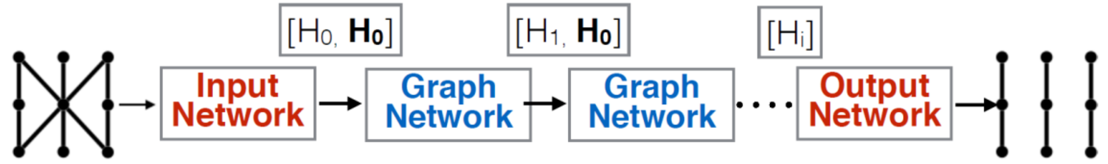
score < cut: fake



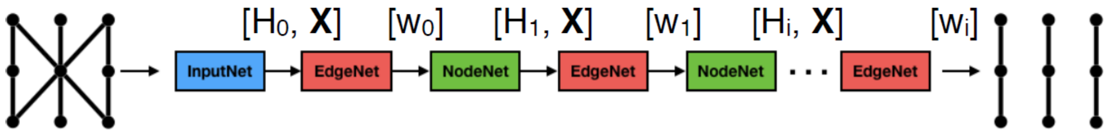
Passing information around the graph gives it learning power

- Can make a node "aware" of its neighbours by concatenating the neighbouring hidden features
- Iterating this neighbourhood learning passes information around the graph
- Can be considered a generalisation of a flat CNN convolution

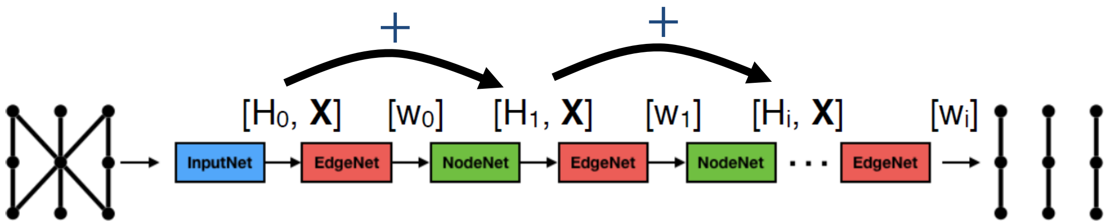
GNN Edge prediction architecture



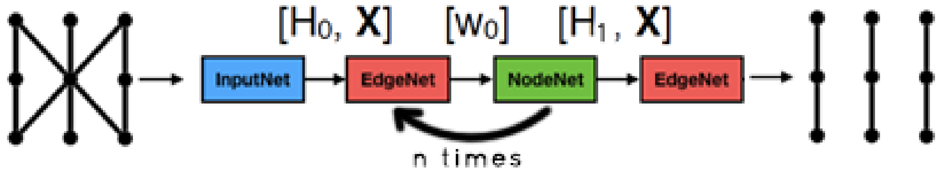
- Message Passing



- Attention Message Passing

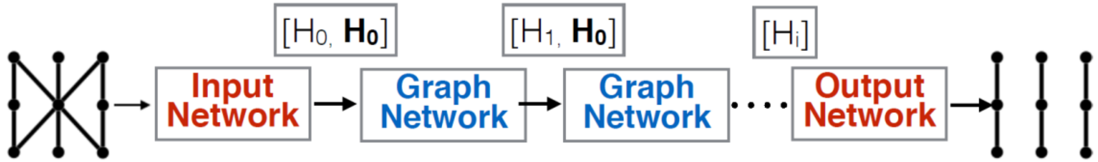


- Attention Message Passing with Residuals

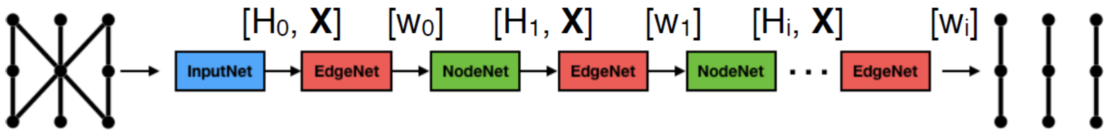


- Attention Message Passing with Recursion

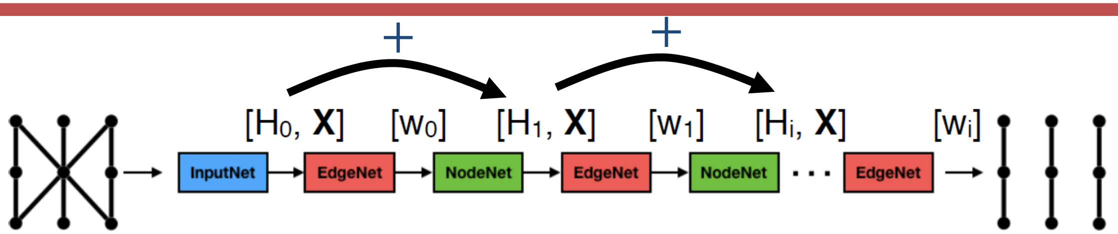
GNN Edge prediction architecture



- Message Passing

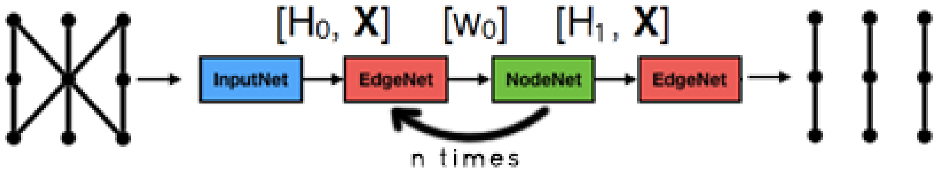


- Attention Message Passing



- Attention Message Passing with Residuals

Have found best efficiency & purity performance.

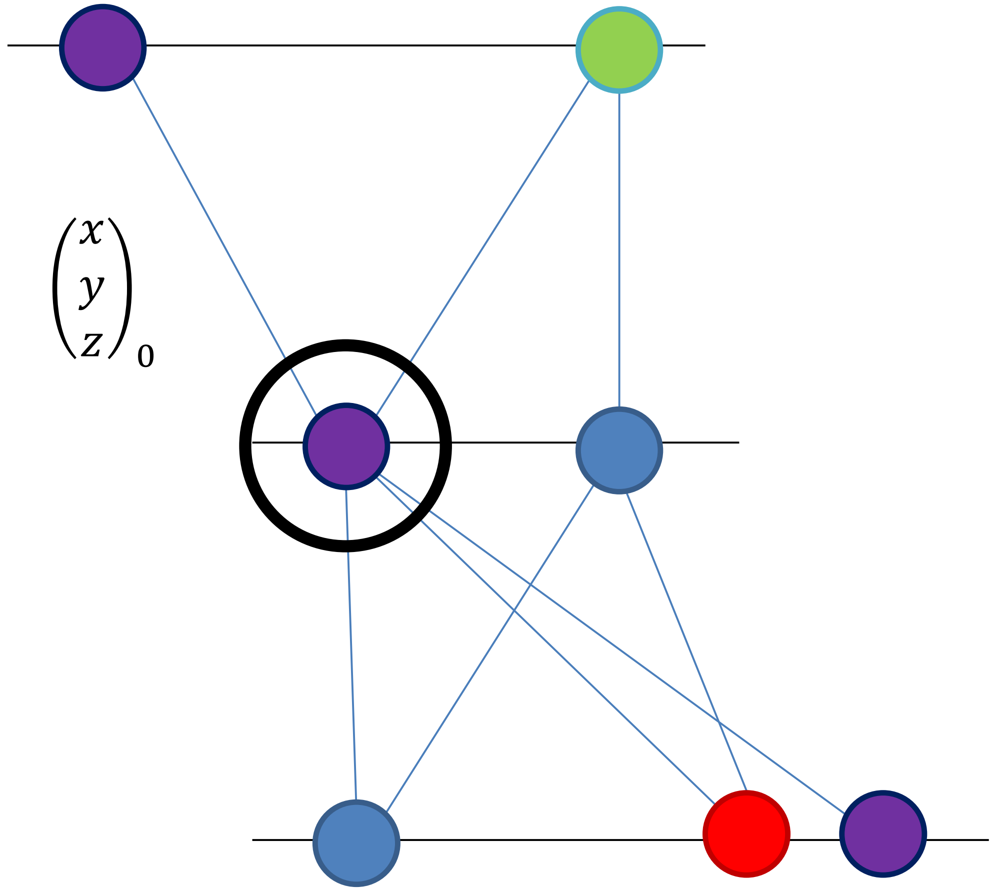


- Attention Message Passing with Recursion

Edge attention architecture

- **Input node features**
- Hidden node features
- Hidden edge features
- Edge score
- Attention aggregation
- New hidden node features
- New hidden edge features
- New edge score

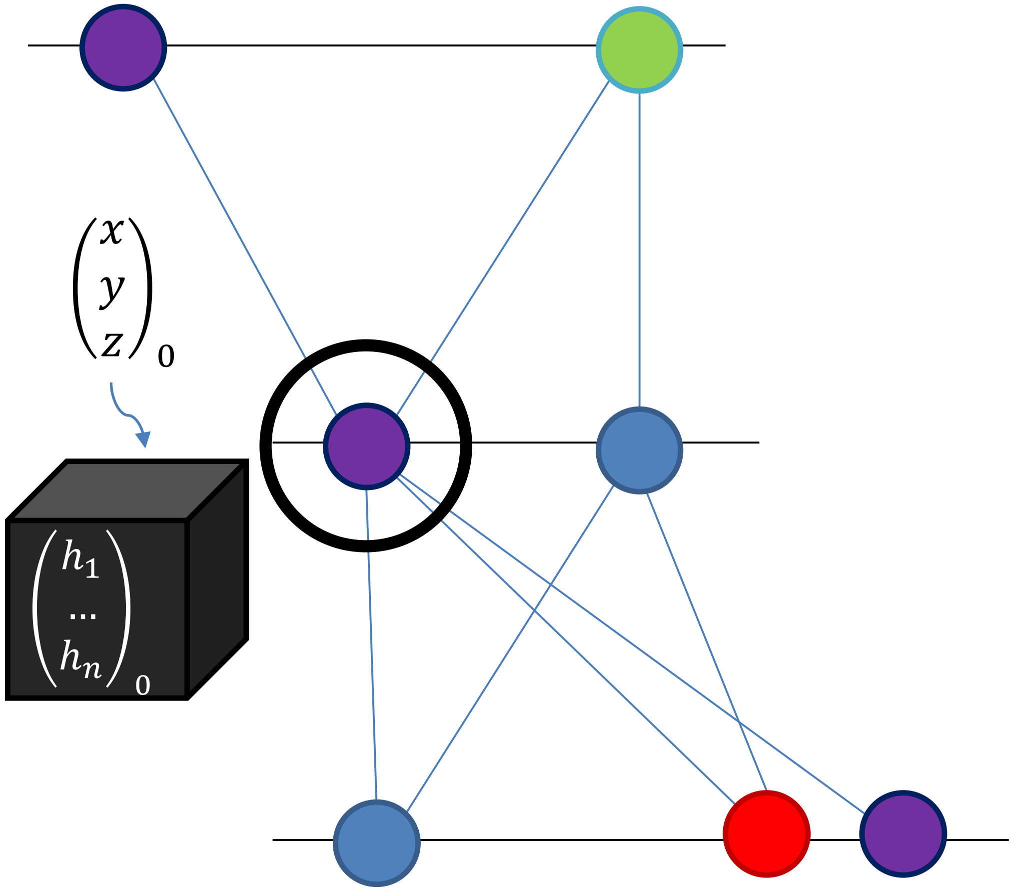
x n iterations
(hyperparameter)



Edge attention architecture

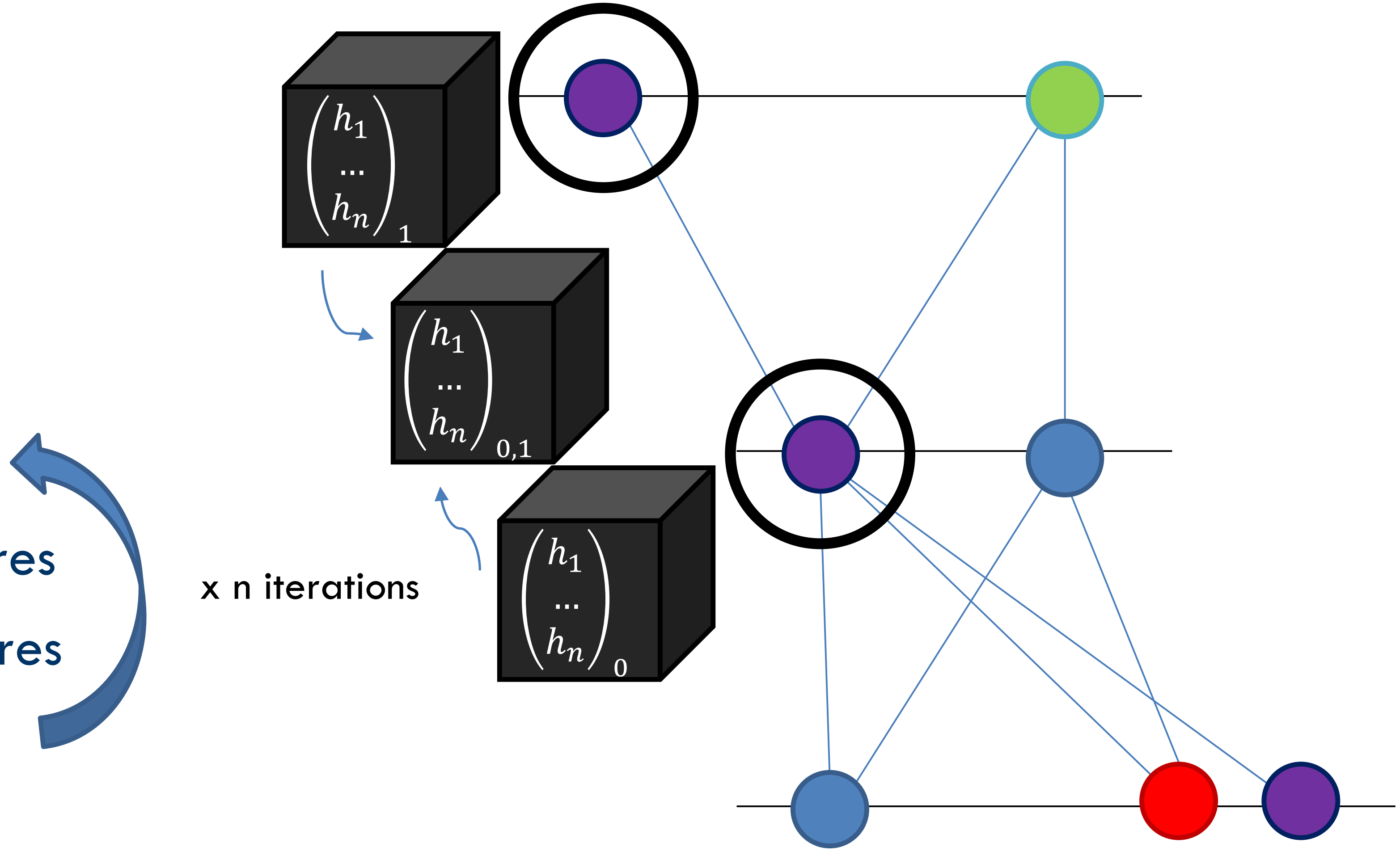
- Input node features
- **Hidden node features**
- Hidden edge features
- Edge score
- Attention aggregation
- New hidden node features
- New hidden edge features
- New edge score

x n iterations



Edge attention architecture

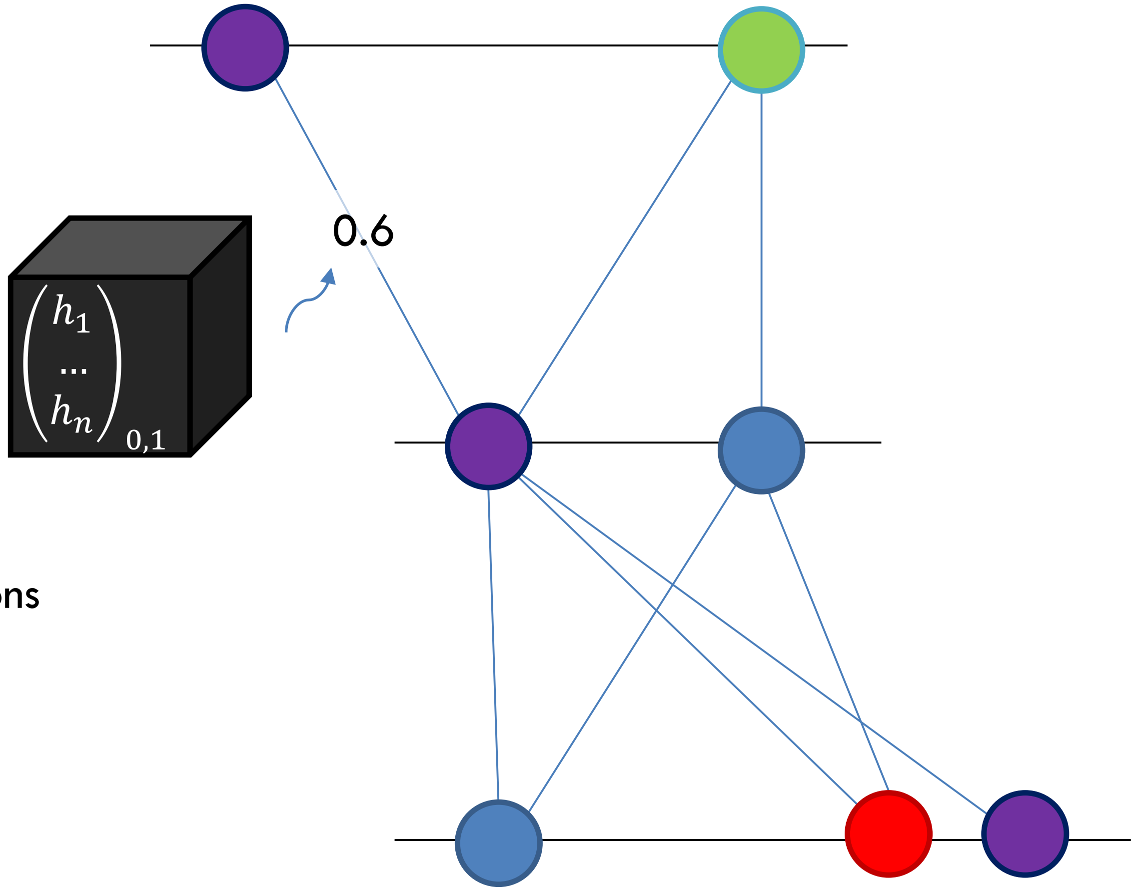
- Input node features
- Hidden node features
- **Hidden edge features**
- Edge score
- Attention aggregation
- New hidden node features
- New hidden edge features
- New edge score



Edge attention architecture

- Input node features
- Hidden node features
- Hidden edge features
- **Edge score**
- Attention aggregation
- New hidden node features
- New hidden edge features
- New edge score

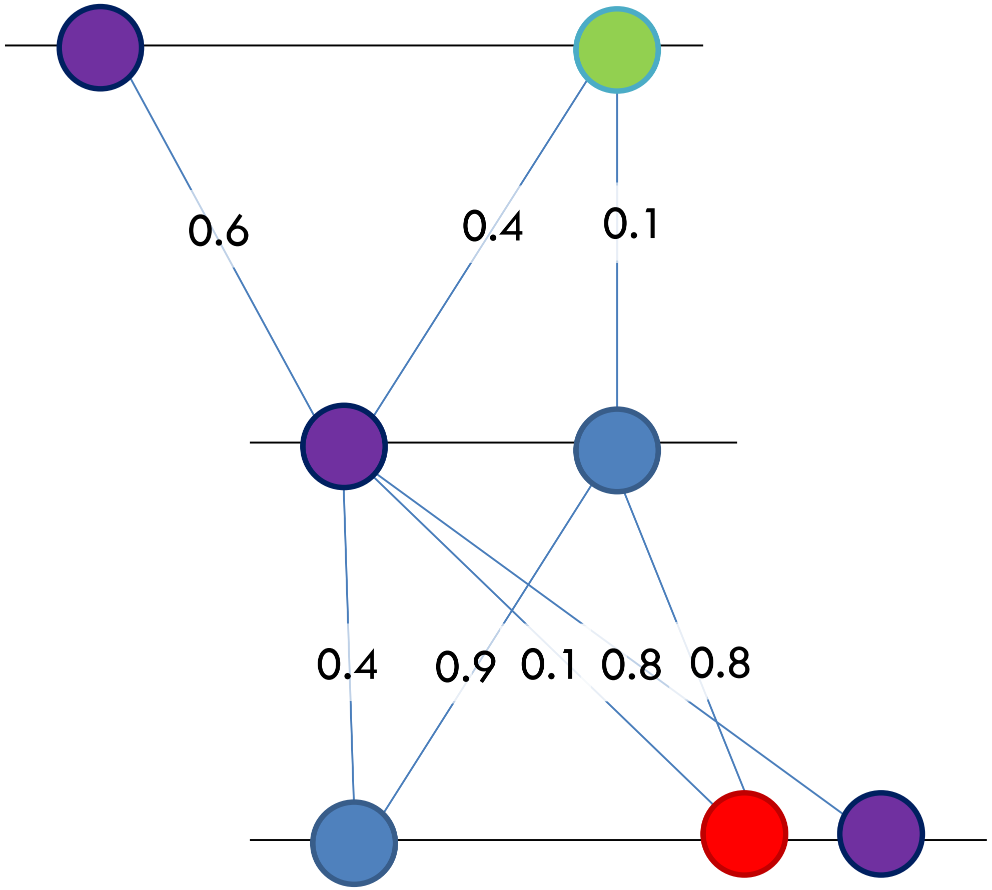
x n iterations



Edge attention architecture

- Input node features
- Hidden node features
- Hidden edge features
- **Edge score**
- Attention aggregation
- New hidden node features
- New hidden edge features
- New edge score

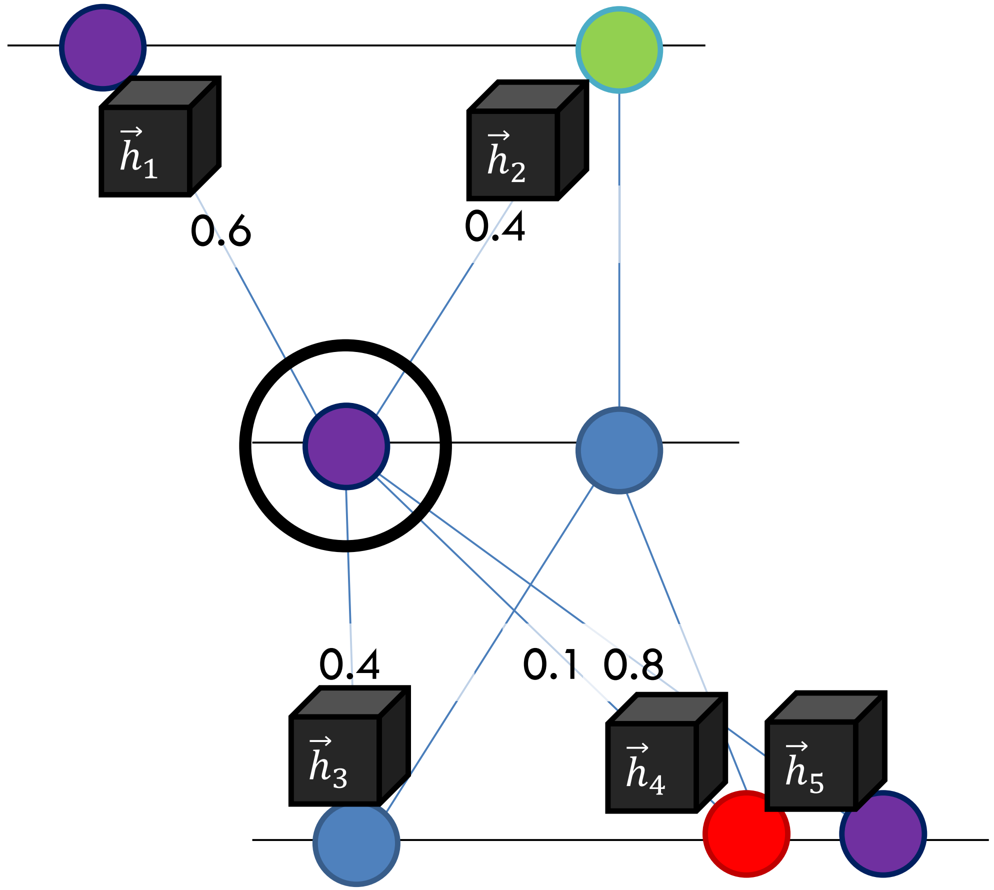
x n iterations



Edge attention architecture

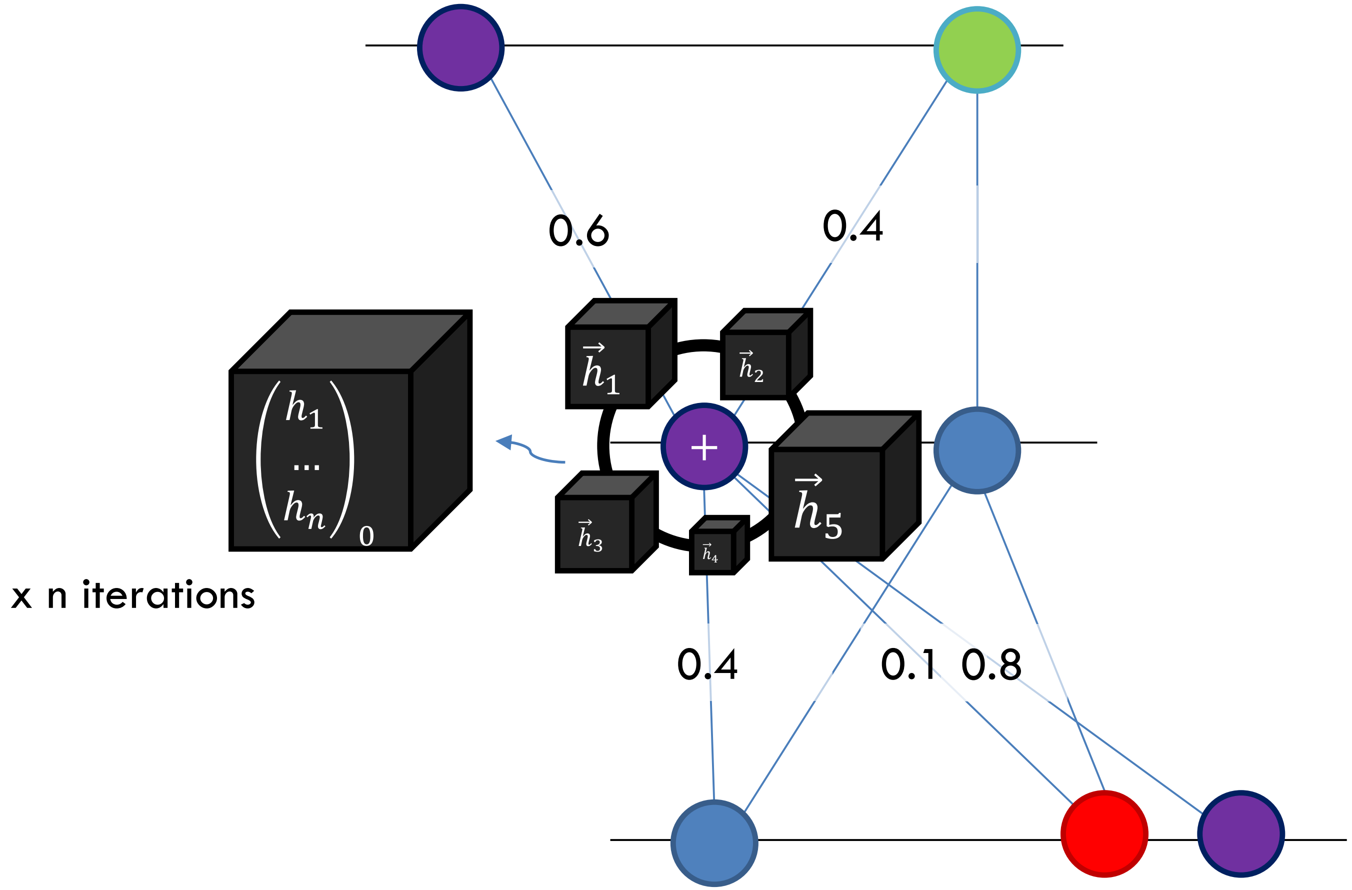
- Input node features
- Hidden node features
- Hidden edge features
- Edge score
- **Attention aggregation**
- New hidden node features
- New hidden edge features
- New edge score

x n iterations



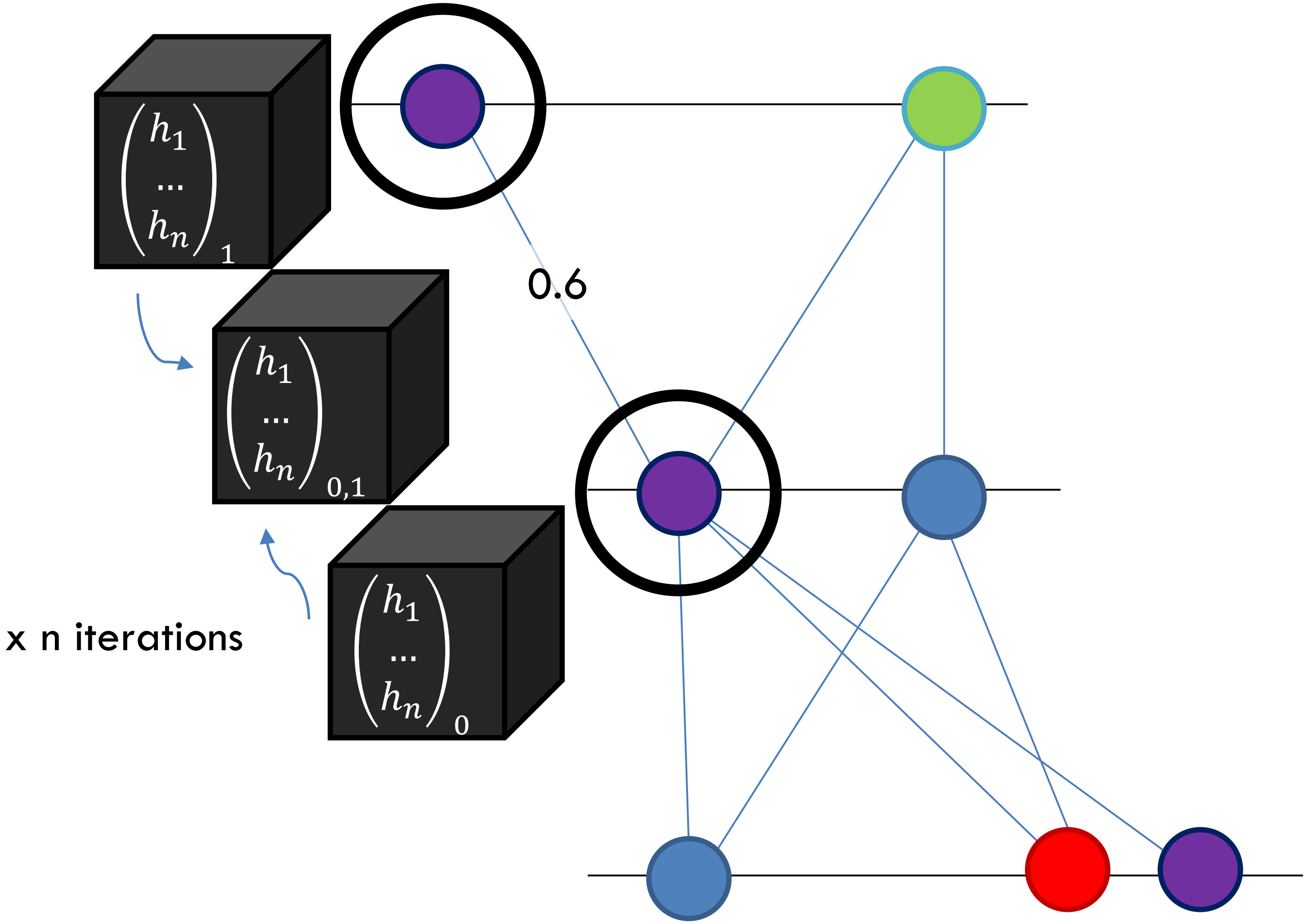
Edge prediction architecture

- Input node features
- Hidden node features
- Hidden edge features
- Edge score
- **Attention aggregation**
- **New hidden node features**
- New hidden edge features
- New edge score



Edge attention architecture

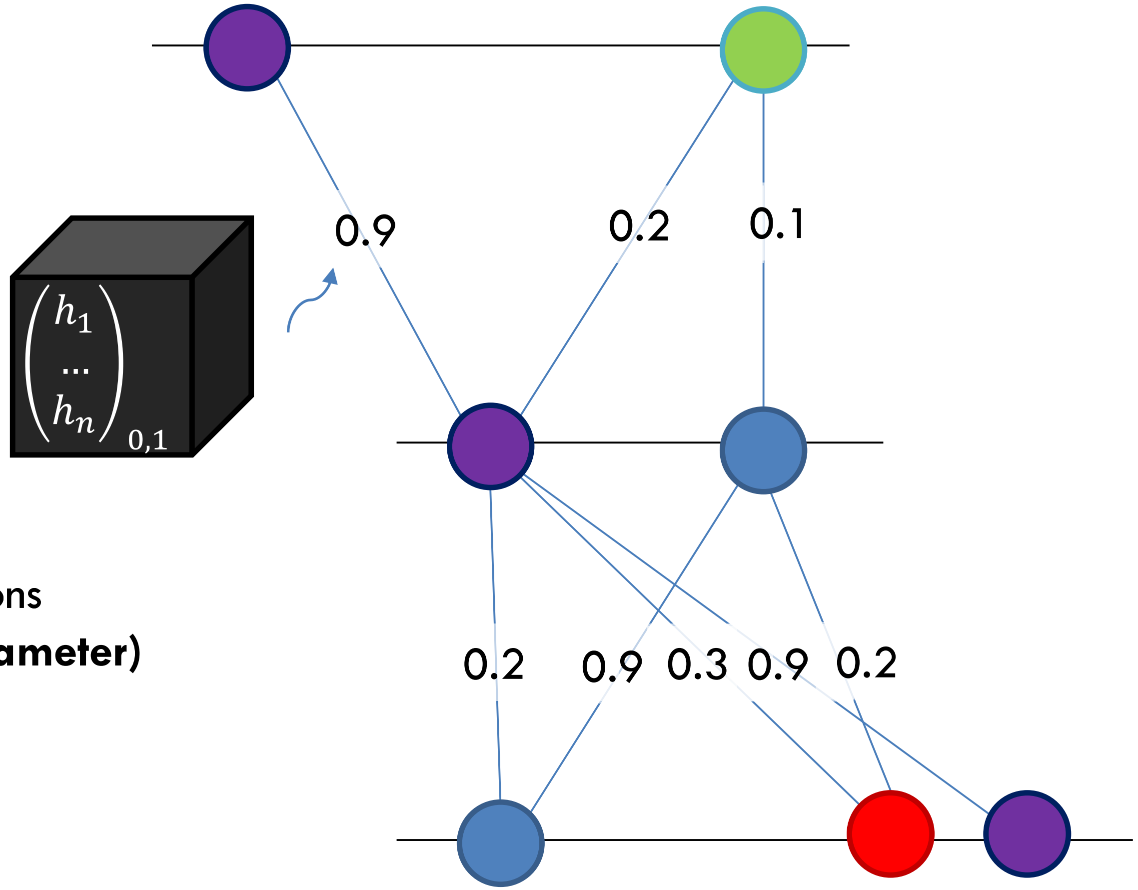
- Input node features
- Hidden node features
- Hidden edge features
- Edge score
- Attention aggregation
- New hidden node features
- **New hidden edge features**
- New edge score



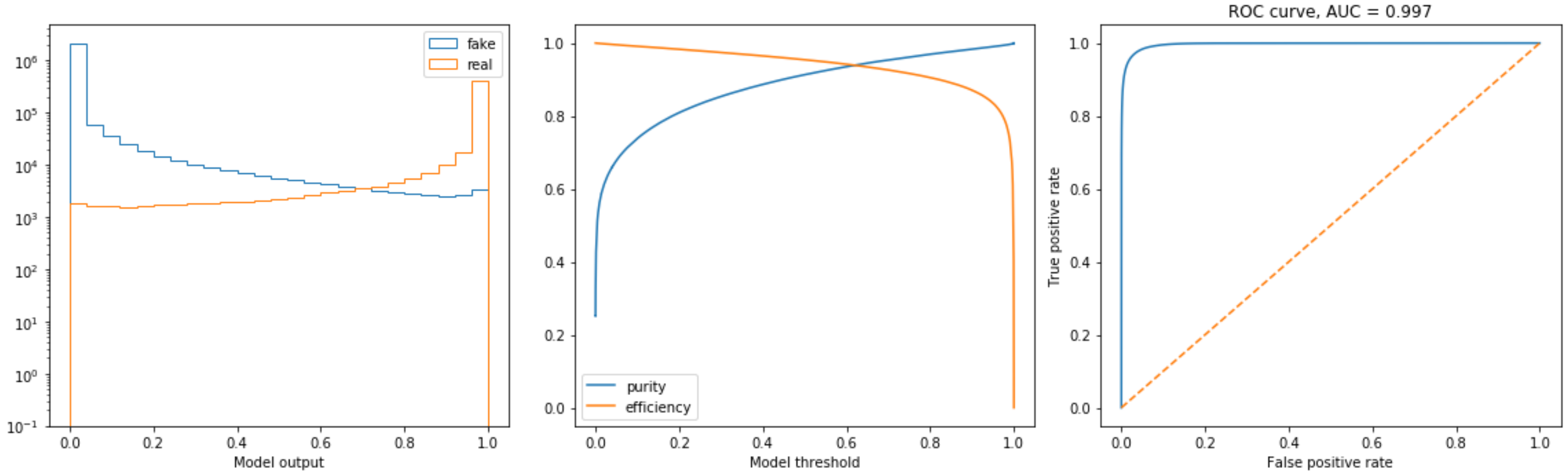
Edge attention architecture

- Input node features
- Hidden node features
- Hidden edge features
- Edge score
- Attention aggregation
- New hidden node features
- New hidden edge features
- **New edge score**

x n iterations
(hyperparameter)



Doublet GNN Performance

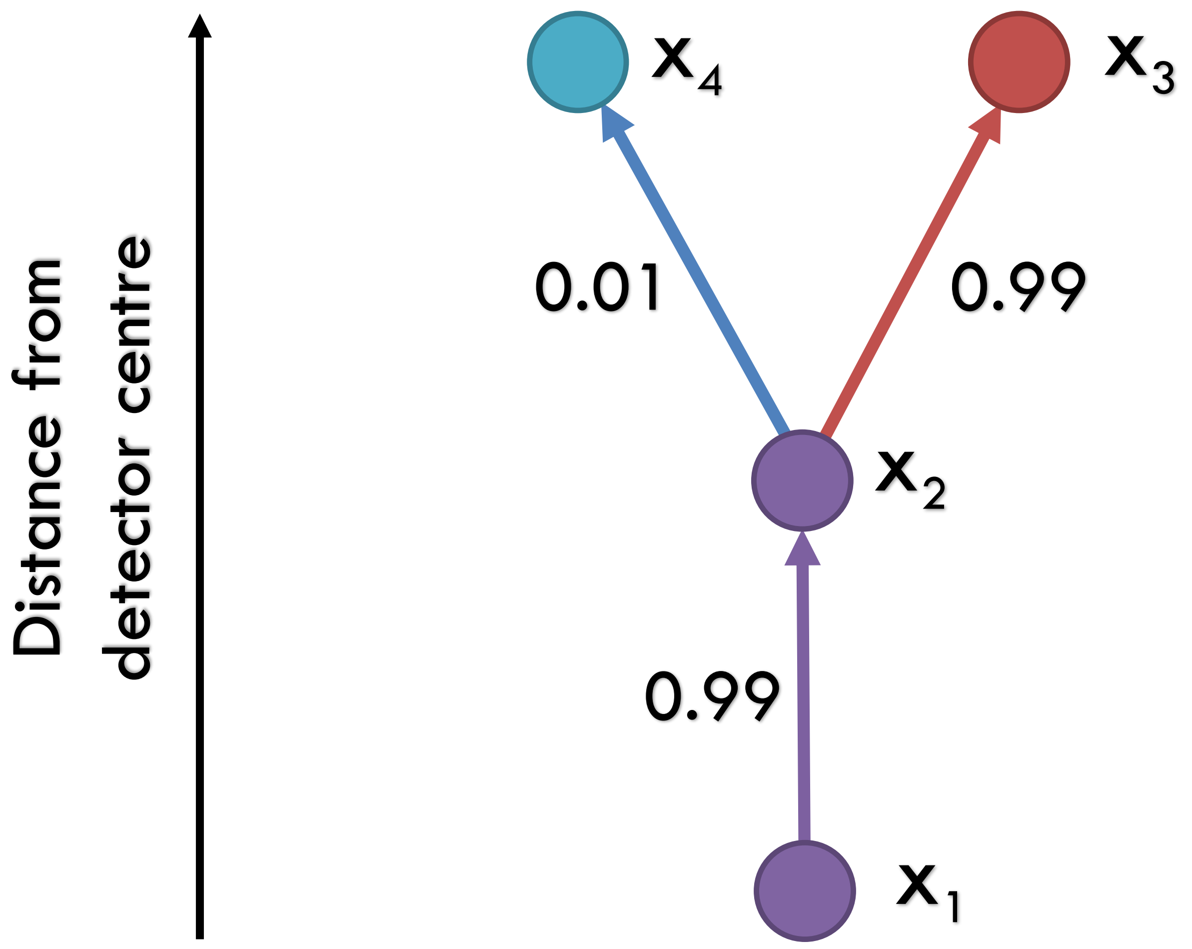


Threshold	0.5	0.8
Accuracy	0.9761	0.9784
Purity	0.9133	0.9694
Efficiency	0.9542	0.9052

Two points to keep in mind

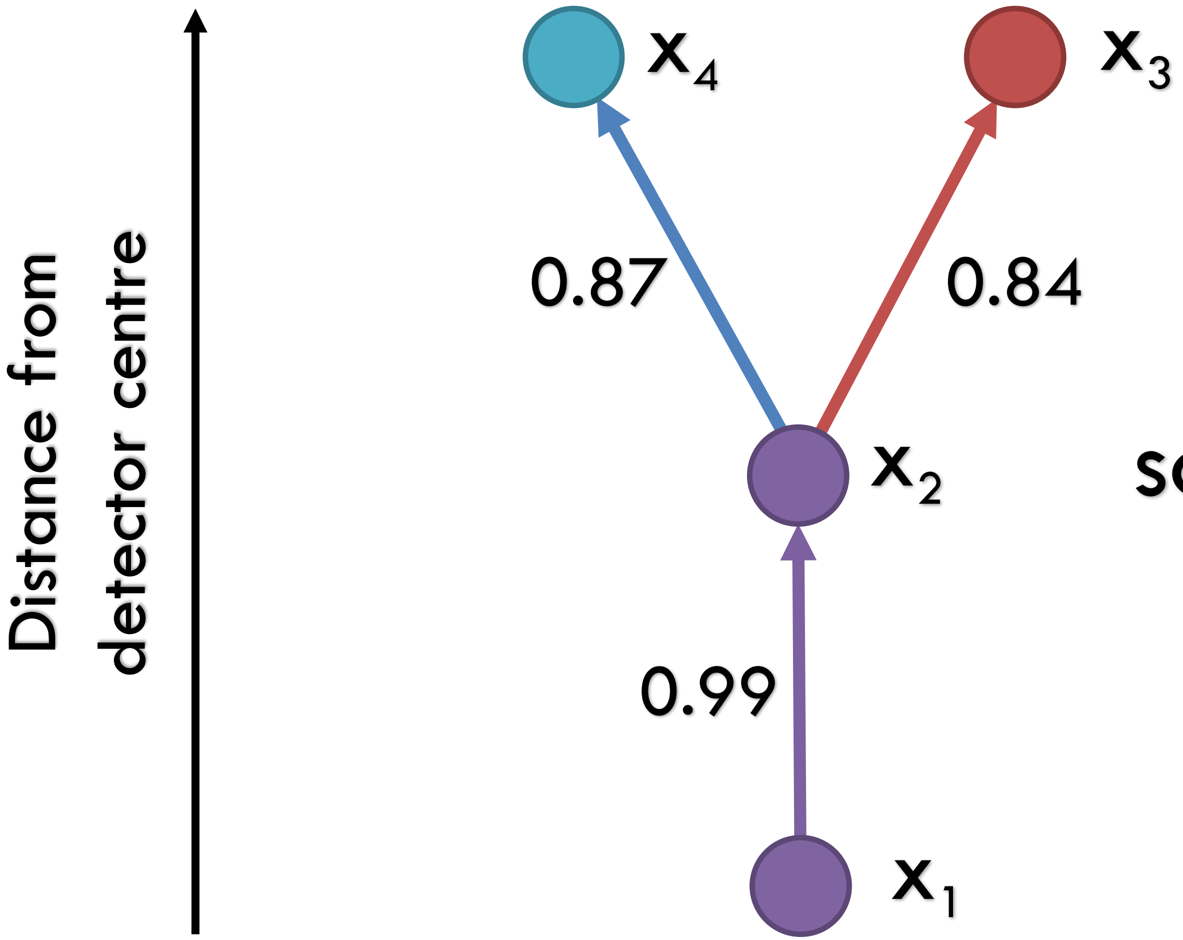
- In the past, graphs have been constructed with a heuristic procedure that had much lower efficiency than the learned embedding. This GNN is classifying a ~ 96% efficient doublet dataset
- These metrics are not the end product: we use the scores of the doublets to create triplets without losing efficiency

Why not simply join together our doublet predictions?



Pretty easy decision

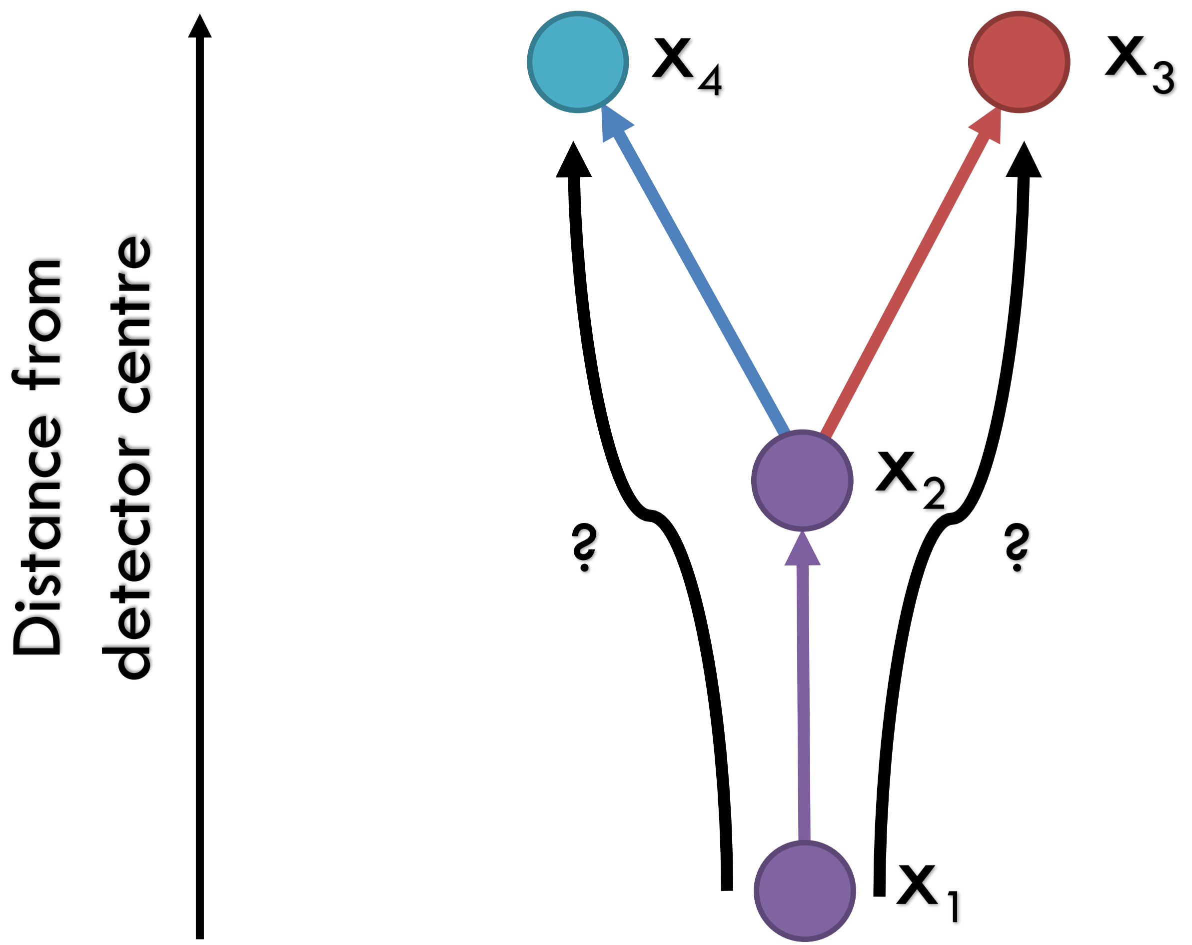
Doublet choice can be ambiguous



Not so easy…

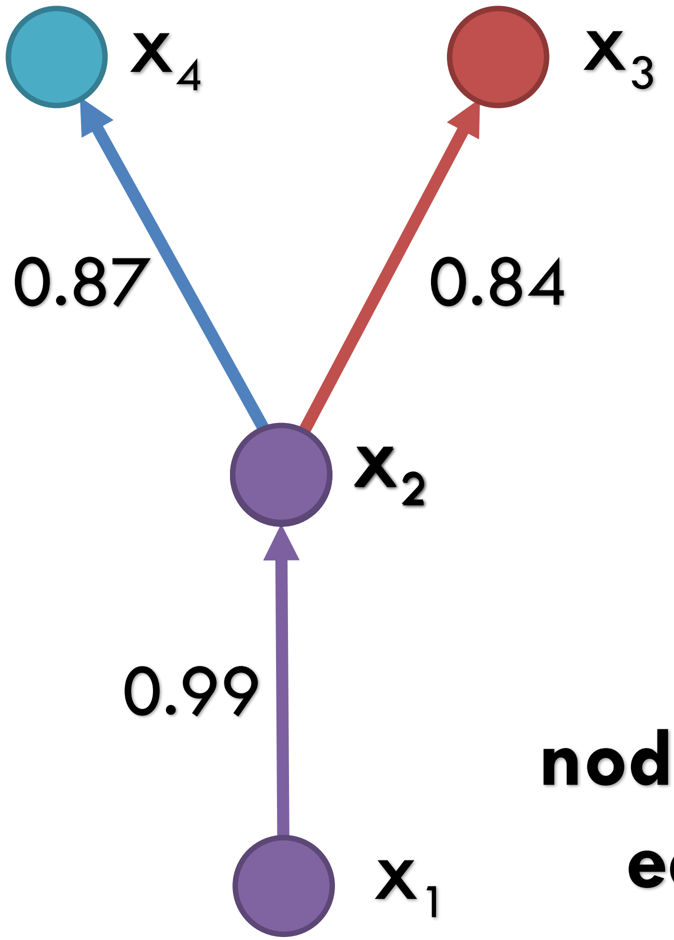
so teach the network how to combine

But a GNN doesn't know about "triplets"



A GNN only knows about nodes and edge

Moving to a "doublet graph" gives us back GNN power

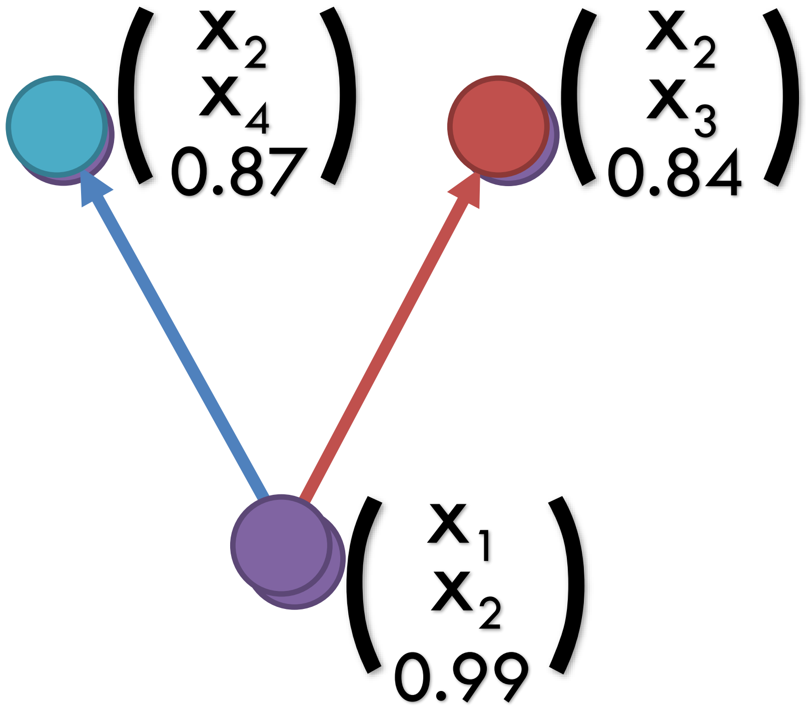


Now…

nodes represent **doublets**,

edges represent **triplets**

Moving to a “doublet graph” gives us back GNN power

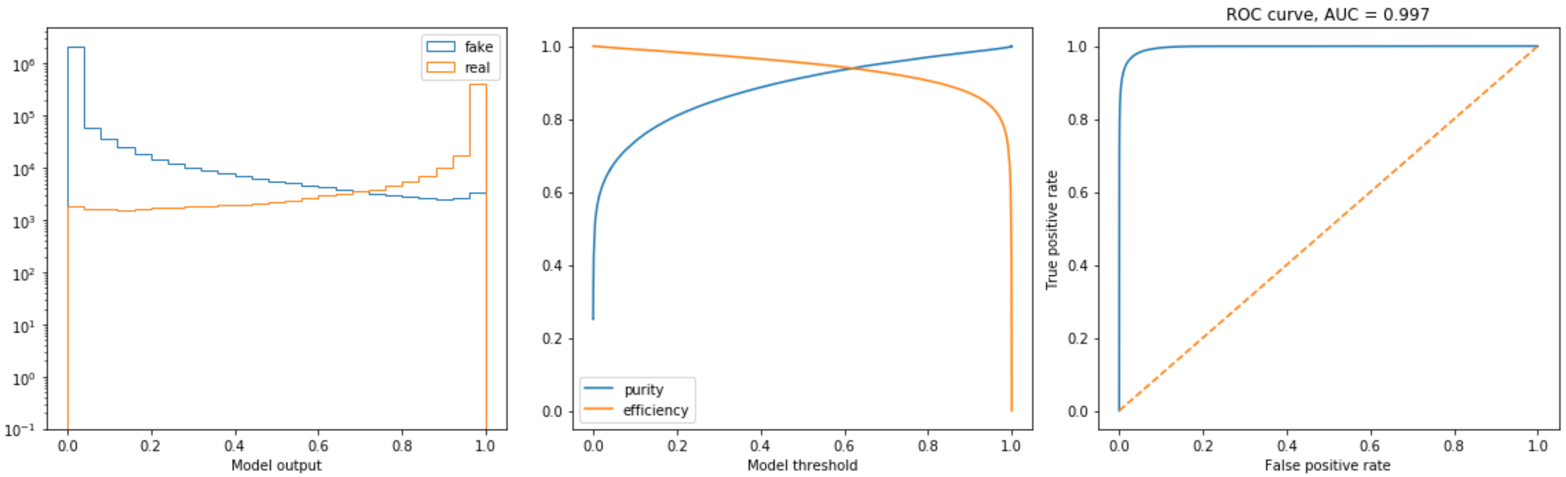


Now…

nodes represent **doublets,**

edges represent **triplets**

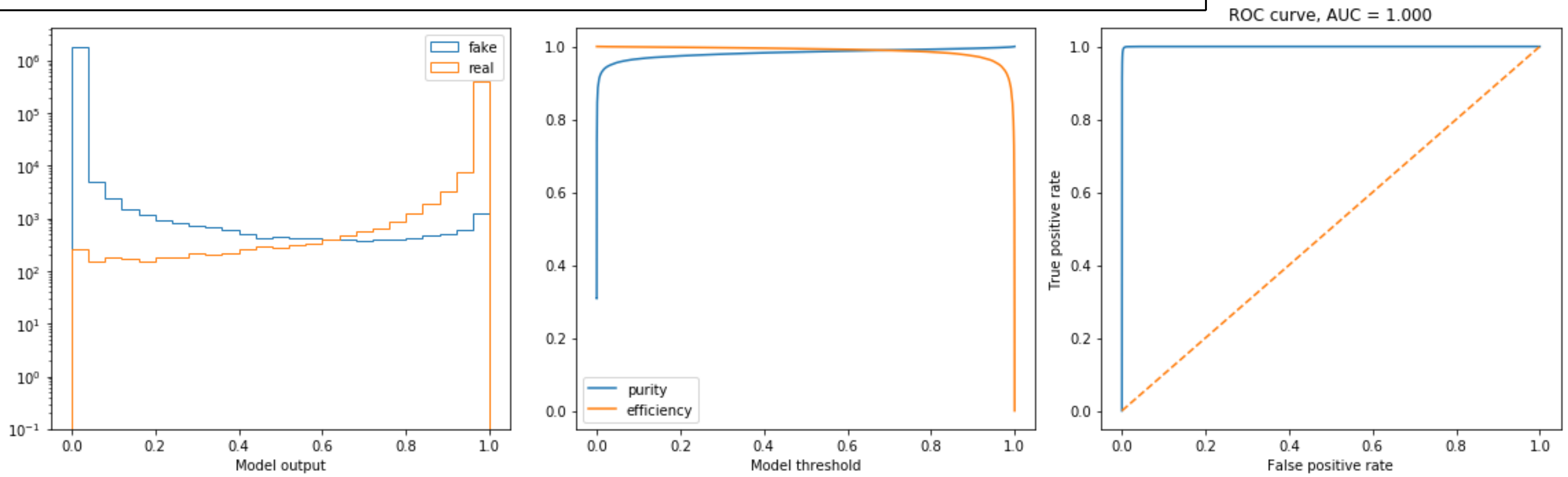
Triplet Propaganda



Threshold	0.5	0.8
Accuracy	0.9761	0.9784
Purity	0.9133	0.9694
Efficiency * relative	0.9542	0.9052

Doublet GNN

Triplet GNN



Threshold	0.5	0.8
Accuracy	0.9960	0.9957
Purity	0.9854	0.9923
Efficiency * relative	0.9939	0.9850

Triplet propaganda

Gold: Unambiguously correct triplet or quadruplet

Other colours: False positive/negative

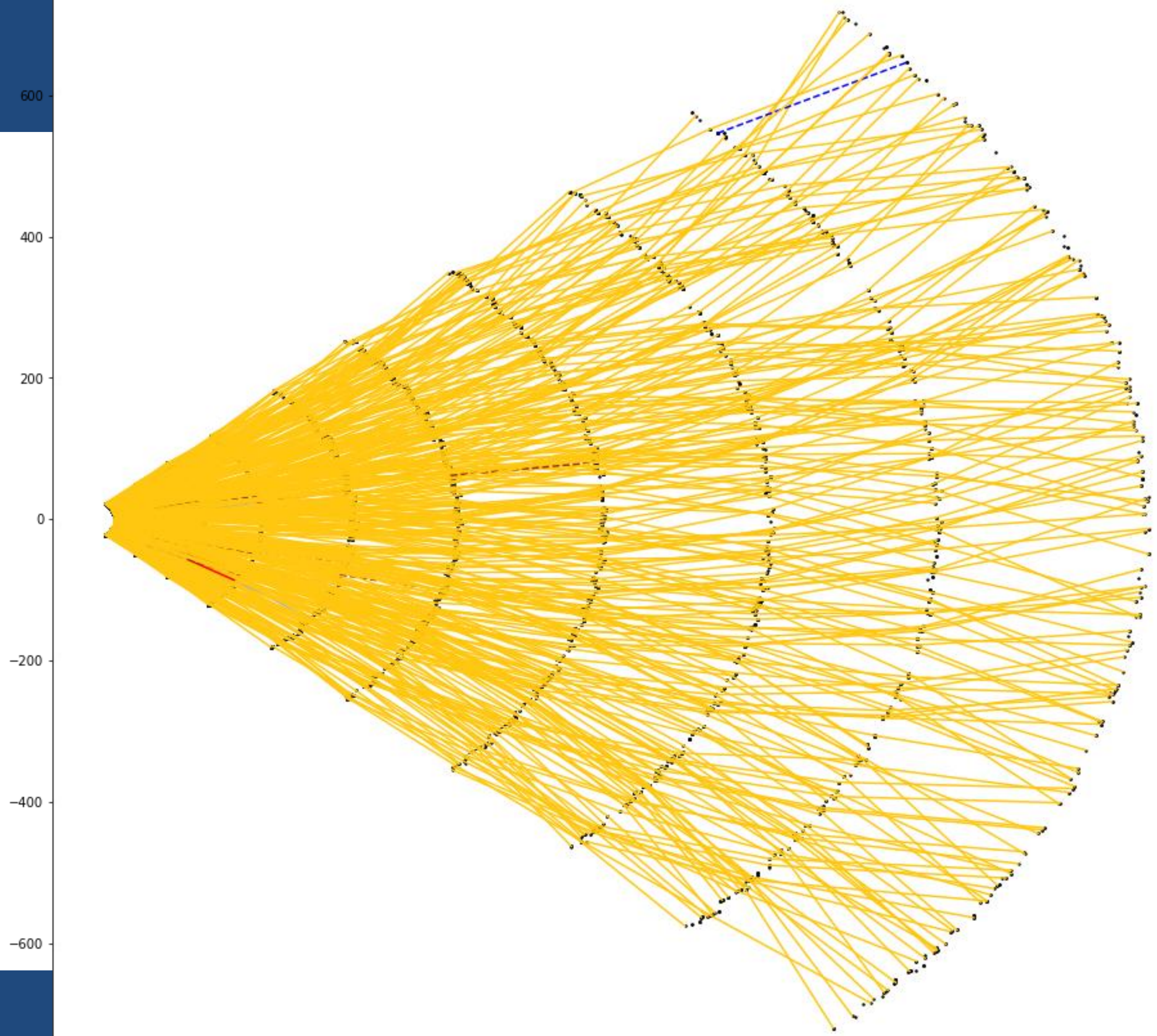
Key:

Silver: Ambiguously correct triplet or quadruplet (i.e. edge shared by correct triplet and false positive triplet)

Bronze dashed: Correct triplet, but missed quadruplet (i.e. edge shared by correct triplet and false negative triplet)

Red: Completely false positive triplet

Blue dashed: Completely false negative triplet



Triplet propaganda

Gold: Unambiguously correct triplet or quadruplet

Other colours: False positive/negative

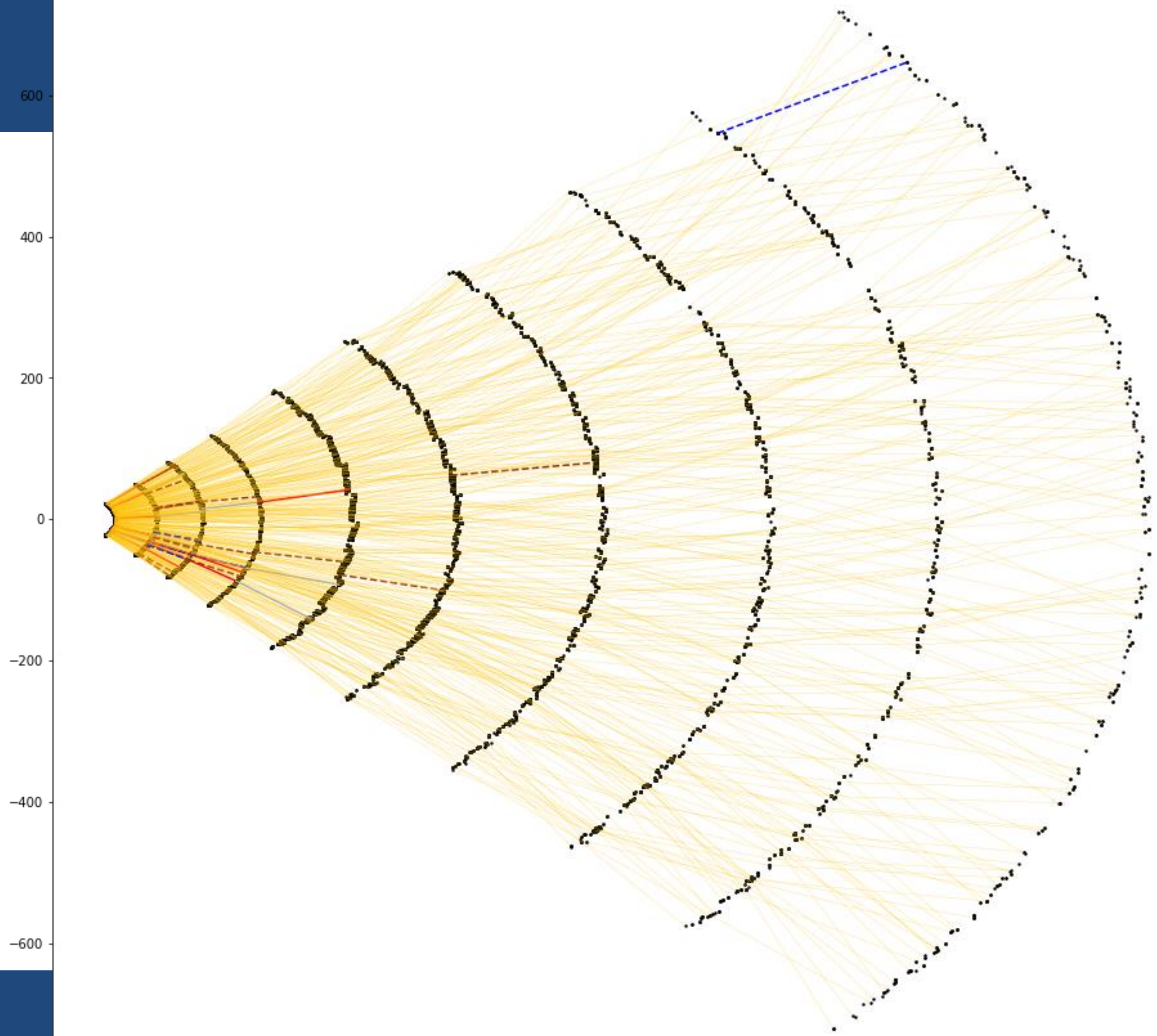
Key:

Silver: Ambiguously correct triplet or quadruplet (i.e. edge shared by correct triplet and false positive triplet)

Bronze dashed: Correct triplet, but missed quadruplet (i.e. edge shared by correct triplet and false negative triplet)

Red: Completely false positive triplet

Blue dashed: Completely false negative triplet



Triplet GNN improves doublet GNN results

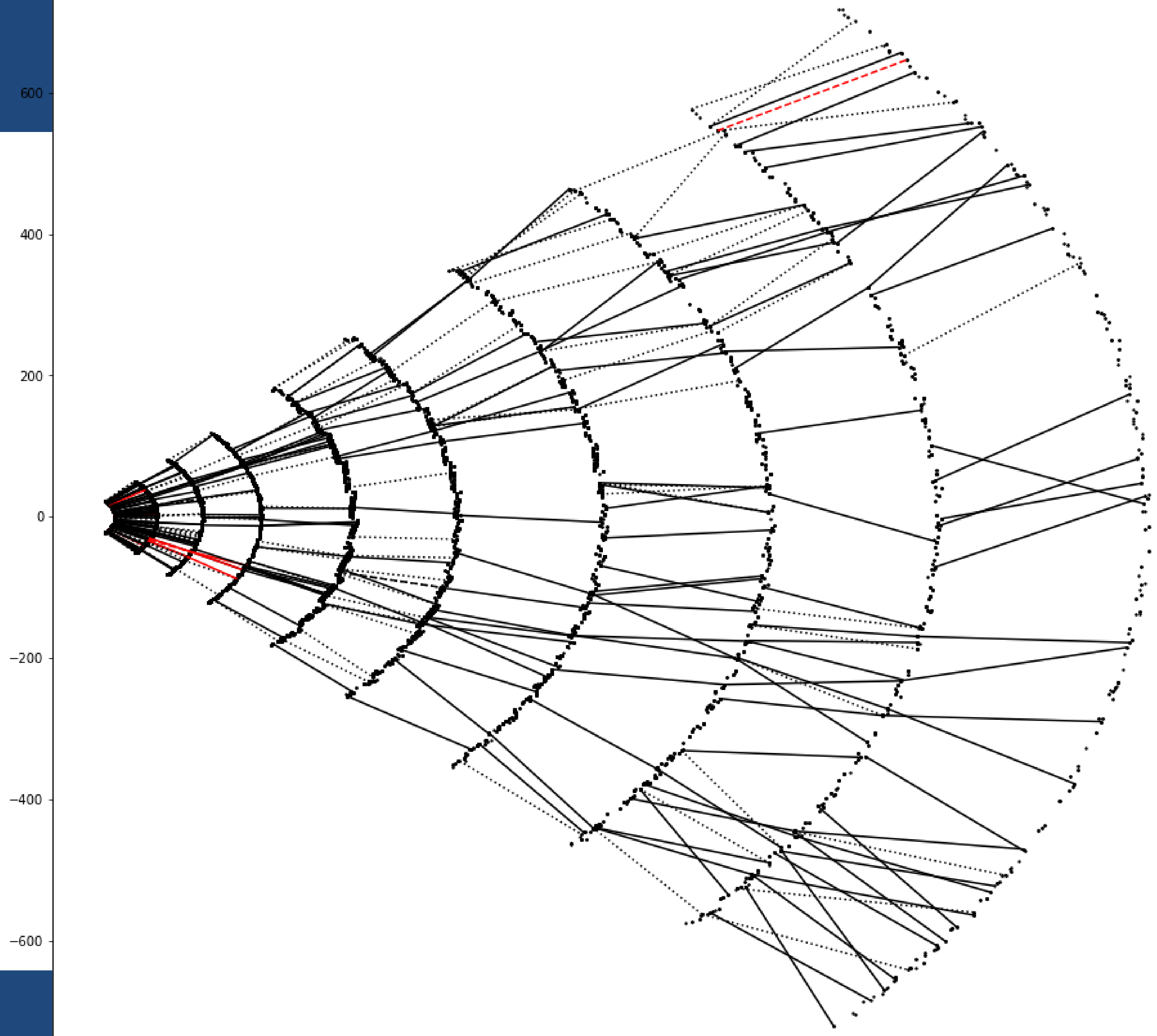
Black: Triplet classifier correctly labelled, doublet classifier mislabelled

Red: Doublet classifier correctly labelled, triplet classifier mislabelled

In this graph, triplet classifier

Fixes 389 edges

Worsens 10 edges



Seeding: Final Performance

Purity: $99.1\% \pm 0.07\%$

Efficiency: $88.6\% \pm 0.19\%$ - This is objective

Inference time: ~ 5 seconds per event per GPU, split between:

- ~ 3 seconds for embedding construction
- ~ 2 seconds for two GNN steps and processing

Seeding: Next Steps

- Direct comparison with ACTS seed generator
- **N-plet GNN**
- The problem is combinatorically increasing graph size

e.g. For TrackML data:

- $O(1{,}000)$ tracks,
- $O(6{,}000)$ hits,
- $O(20{,}000)$ doublets,
- $O(60{,}000)$ triplets

$\Rightarrow$

- Cut doublet input before triplet construction
- Doublet threshold of 0.01 retains 99% efficiency
- Reduces doublets $O(20{,}000) \rightarrow O(6{,}000)$
- We thus have a sustainable process to N-plet GNN

Track Labelling

GOAL

Given a classified doublet and/or triplet graph,

use edge scores to group likely nodes into tracks

and label with unique identifier.

DBSCAN on a Graph

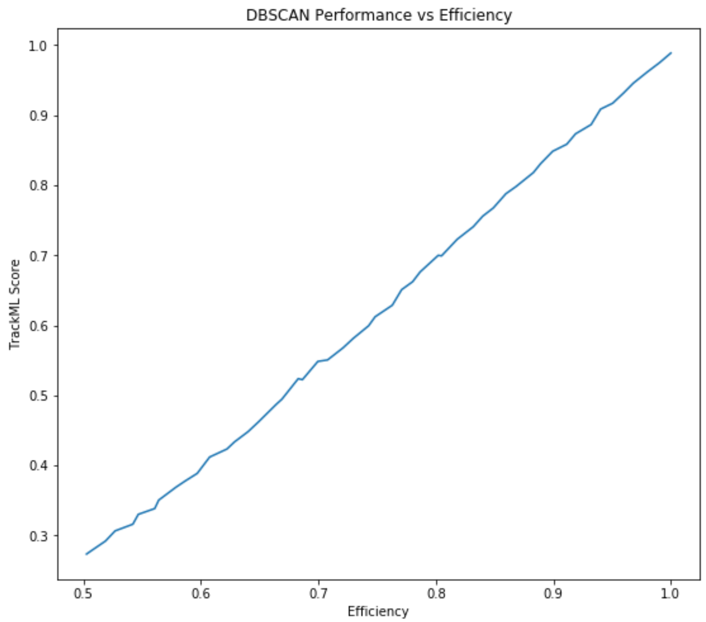
- DBSCAN typically calculates a distance metric and clusters based on neighbourhood density
- Feed the edge scores e_{ij} as a *precomputed,* sparse, metric matrix, with each distance element given by

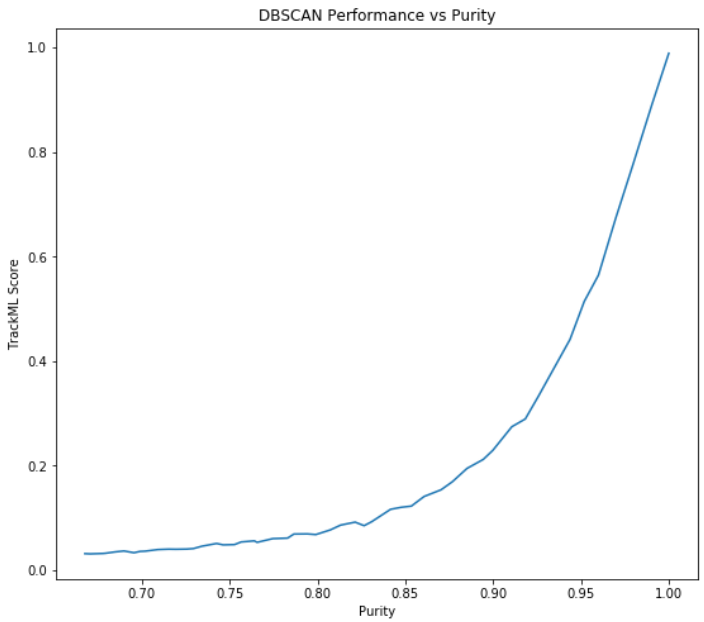
$$d_{ij} = 1 - e_{ij}$$

- Fill out sparse matrix to ensure it is diagonal, i.e. undirected. A directed graph does not perform well with DBSCAN.

DBSCAN Performance

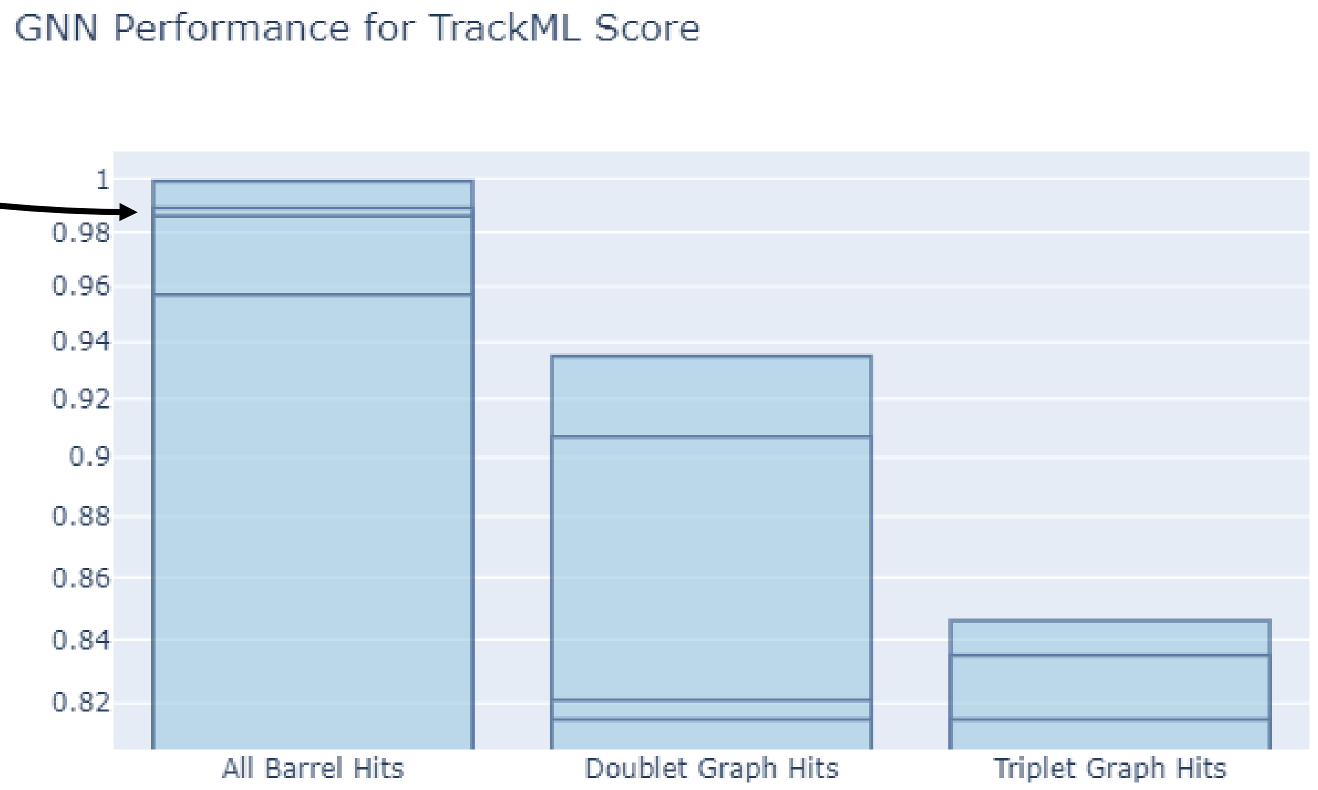
- We can construct a “truth graph” from TrackML data, where every hit is connected to hits of a shared track in adjacent layers, with a high score (e.g. 0.99), and randomly connected to other hits with a low score (e.g. 0.01)
- We can randomly mislabel true edges to reduce efficiency, or mislabel fake edges to reduce purity
- We see linear reduction in TrackML score against efficiency
- Exponential reduction in TrackML score against purity





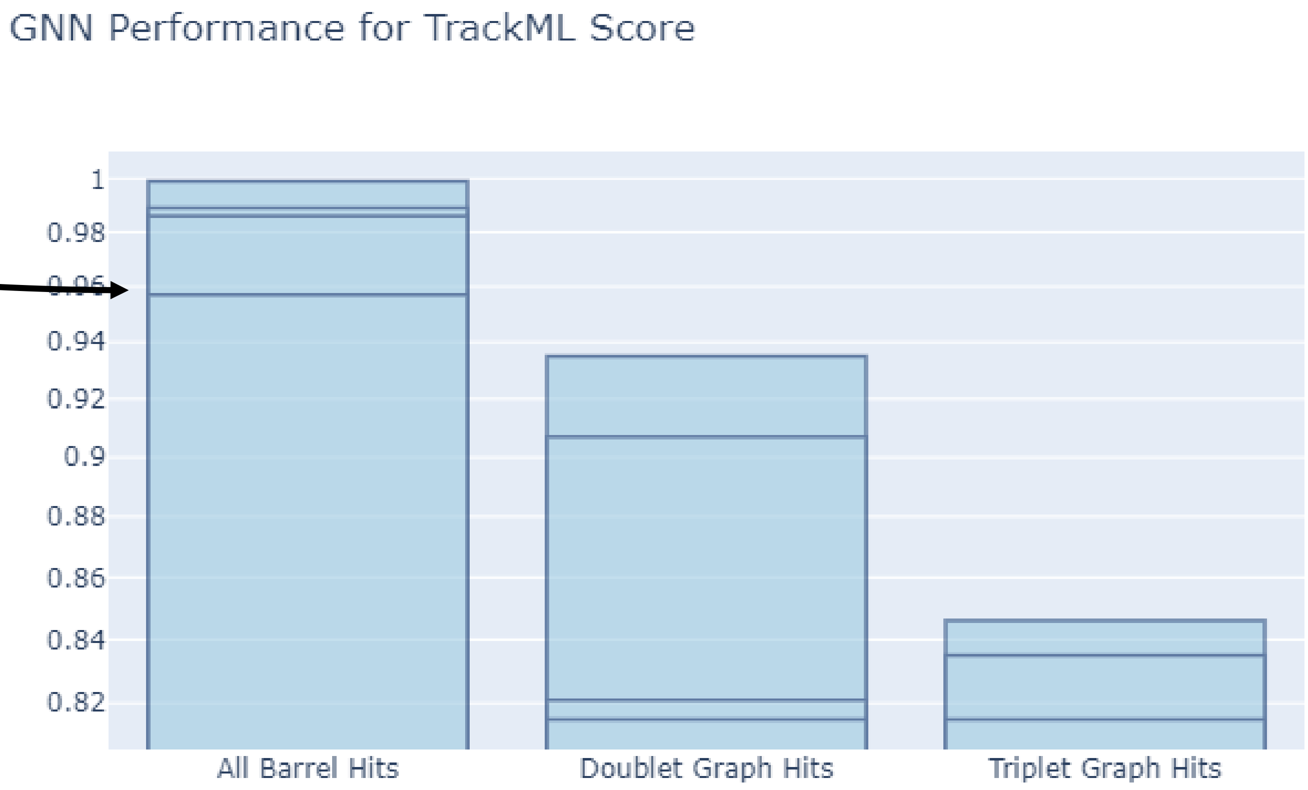
GNN TrackML Score Performances

- DBSCAN on truth graph 0.989



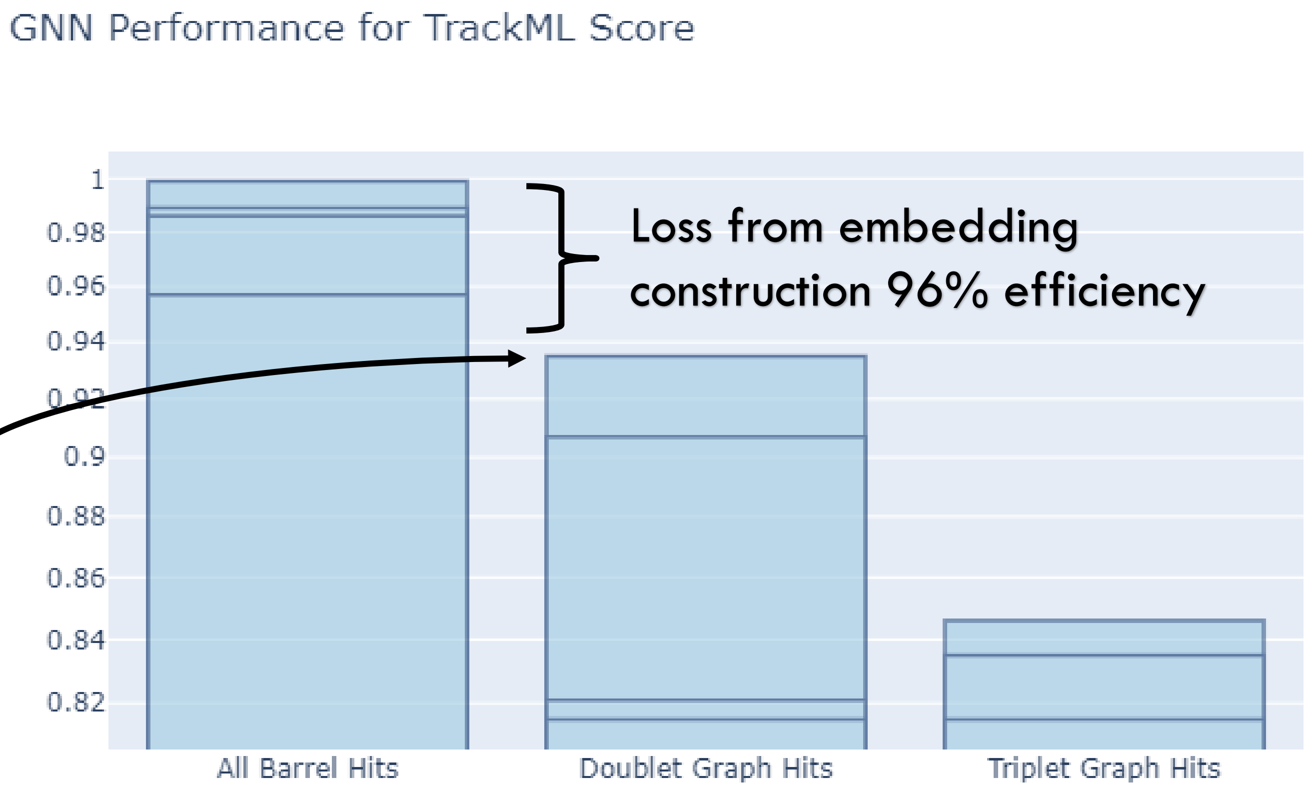
GNN TrackML Score Performances

- DBSCAN on truth graph
  0.989
- DBSCAN on adjacent-layer truth graph
  0.957



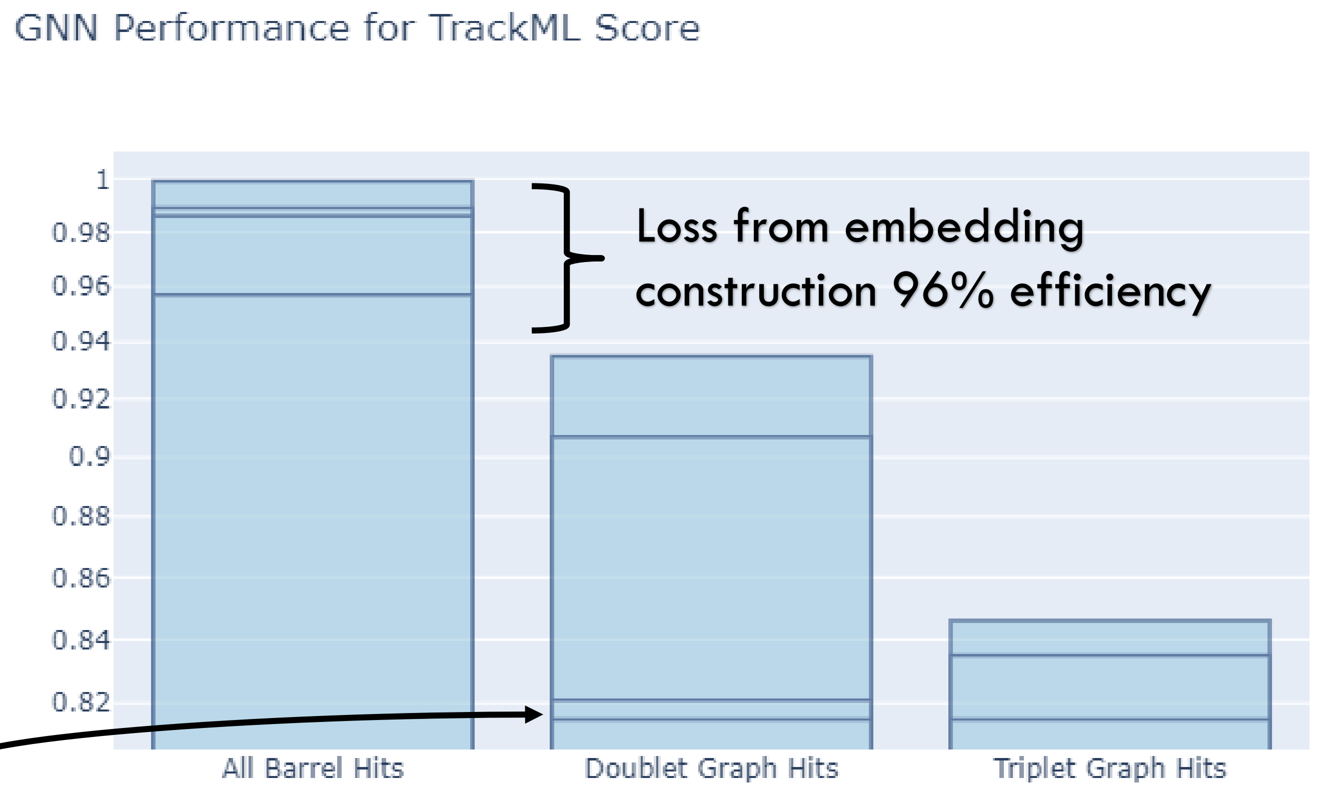
GNN TrackML Score Performances

- DBSCAN on truth graph
  0.989
- DBSCAN on adjacent-layer truth graph
  0.957
- Embedding-constructed doublet hits
  0.935



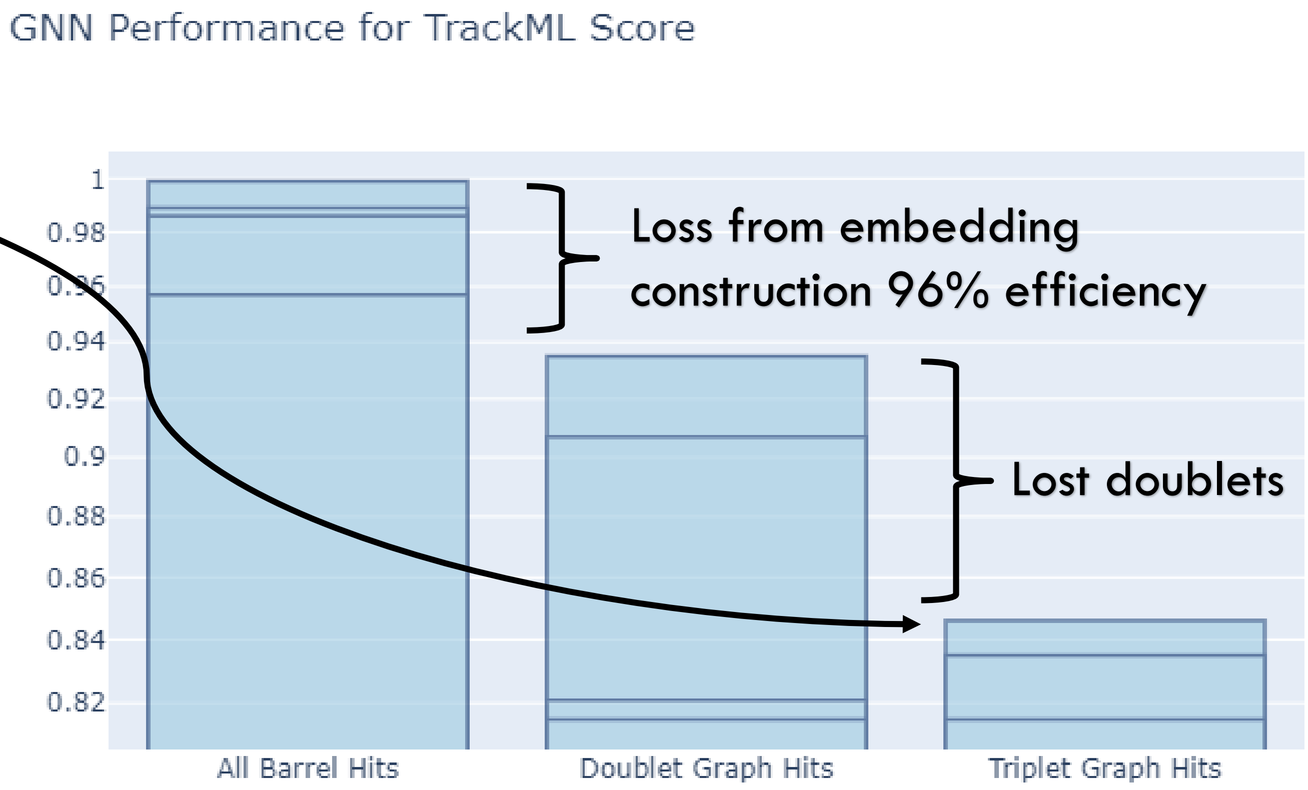
GNN TrackML Score Performances

- DBSCAN on truth graph
  0.989
- DBSCAN on adjacent-layer truth graph
  0.957
- Embedding-constructed doublet graph using truth
  0.935
- DBSCAN on doublet GNN classification
  0.815



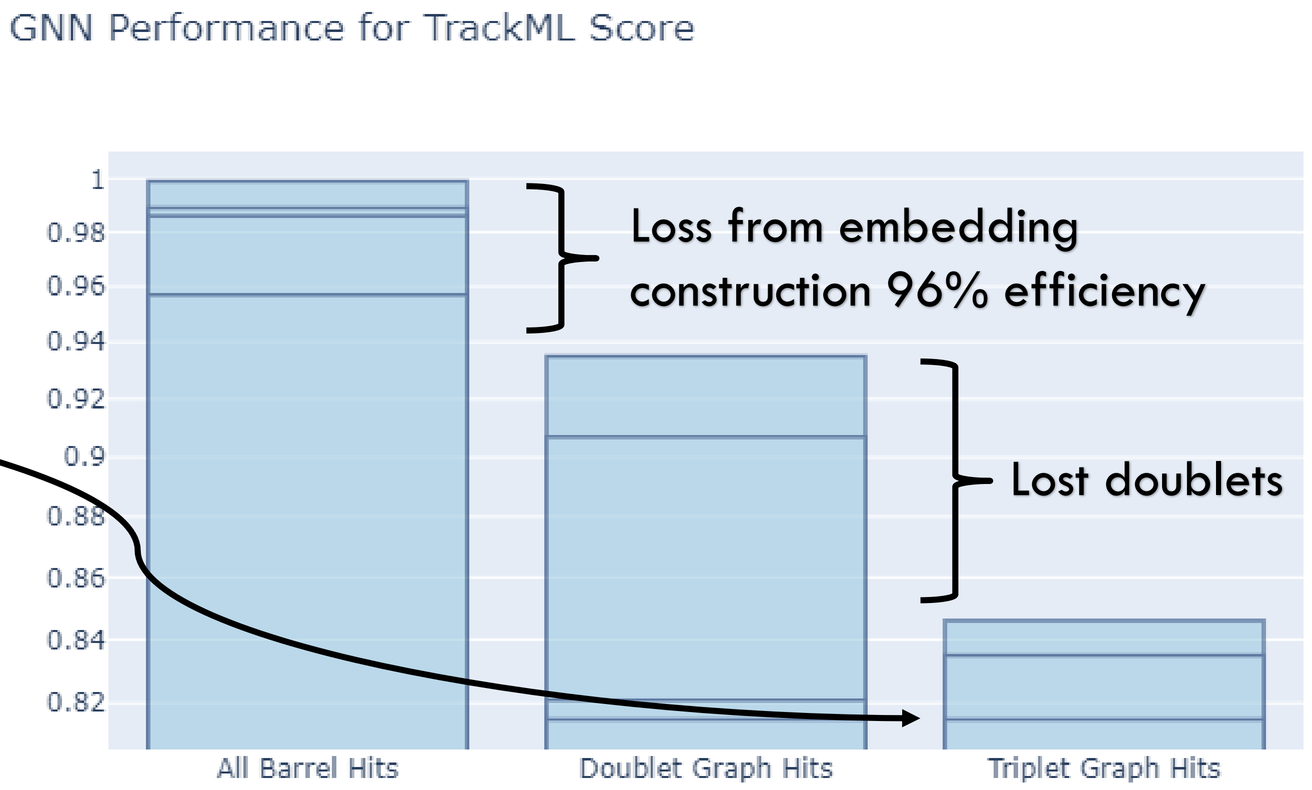
GNN TrackML Score Performances

- Triplet graph constructed from doublet graph (truth) 0.846

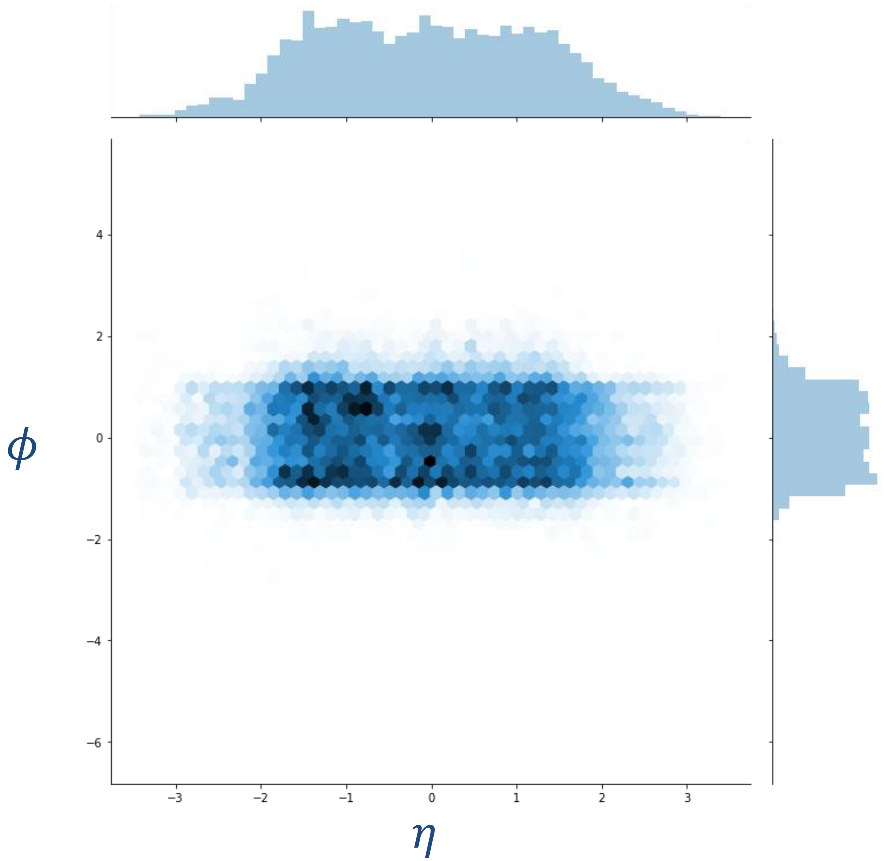


GNN TrackML Score Performances

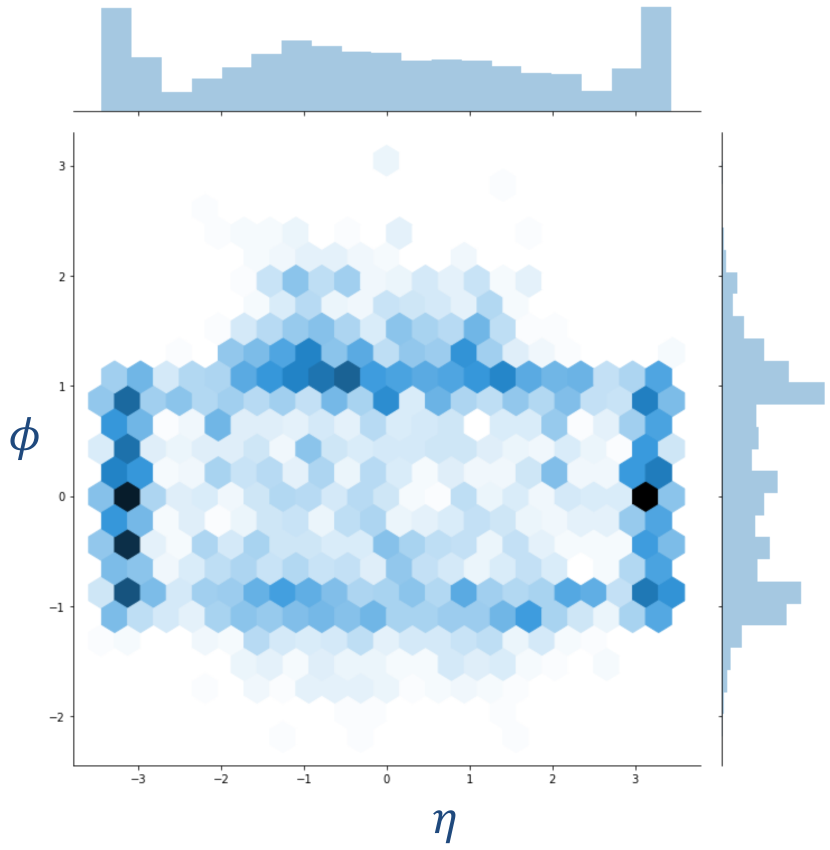
- Triplet graph constructed from doublet graph (truth) 0.846
- DBSCAN on triplet graph from triplet GNN classification 0.815



All Hits

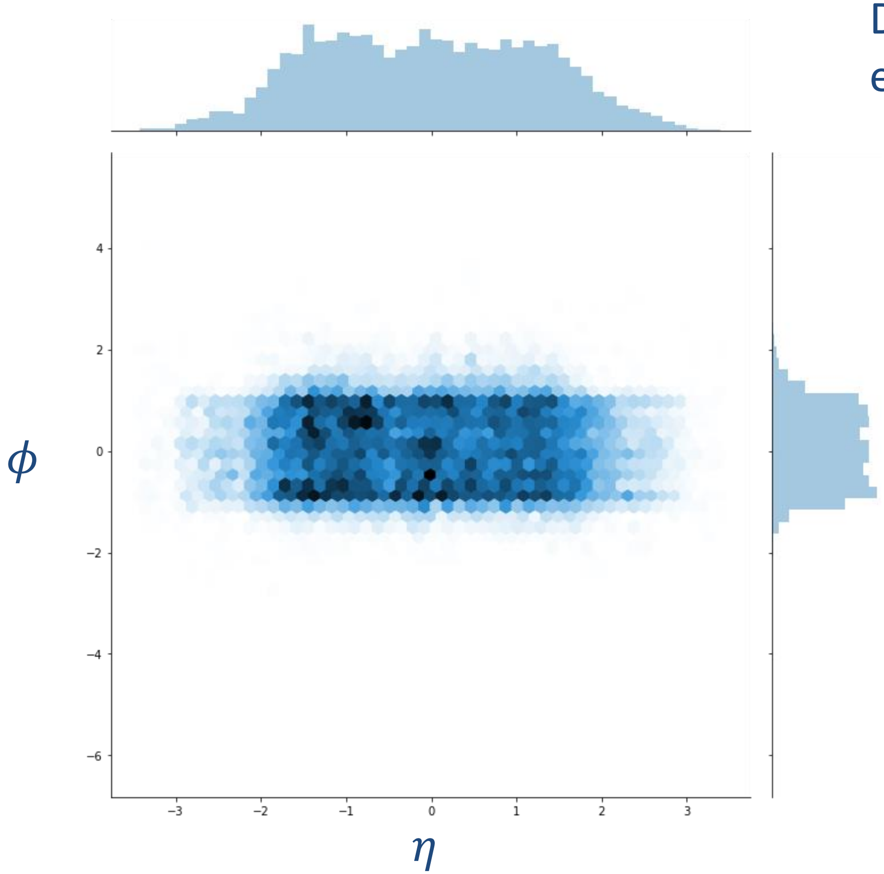


Missing Doublet Hits



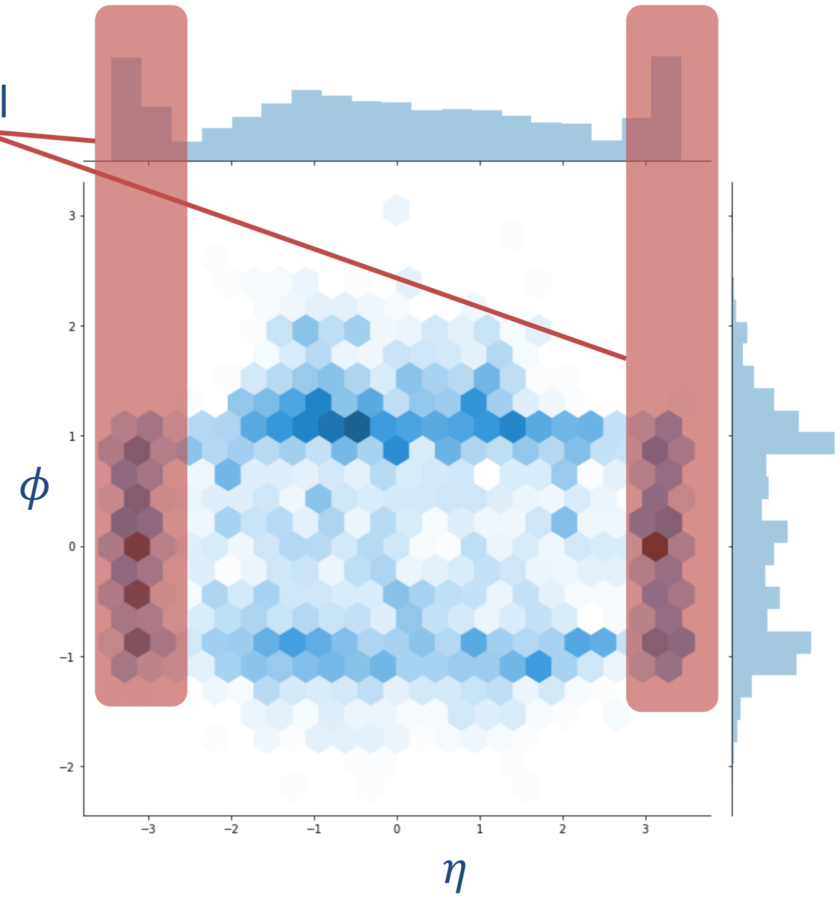
Missing Doublets

All Hits



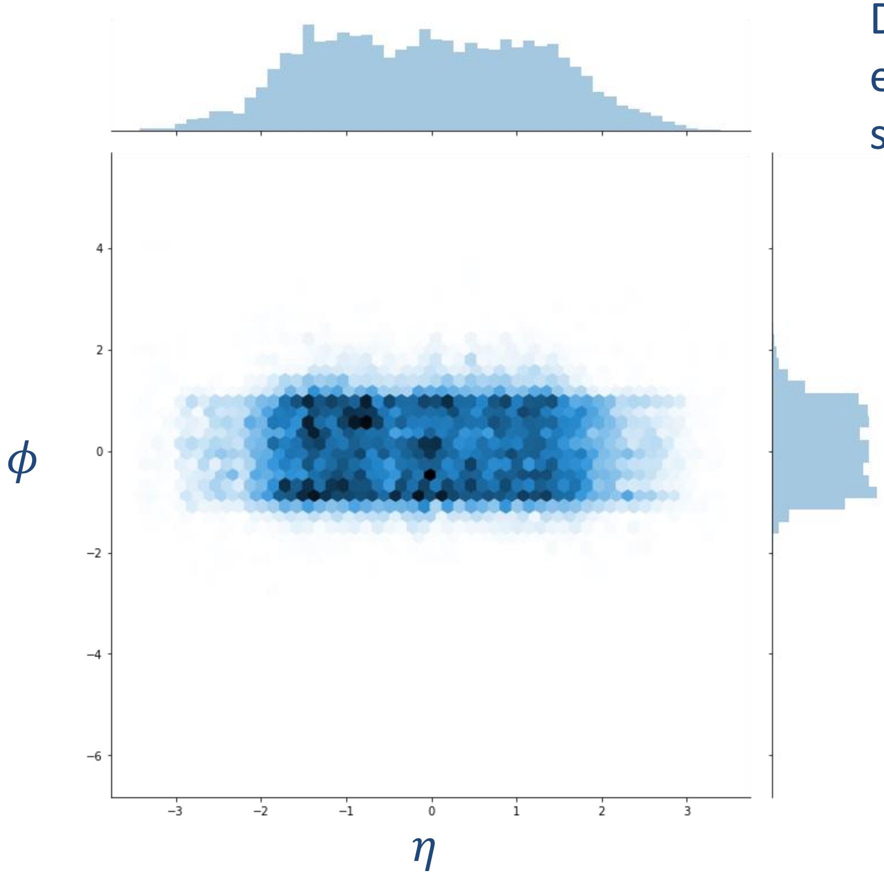
Missing Doublet Hits

Doublets on end of barrel



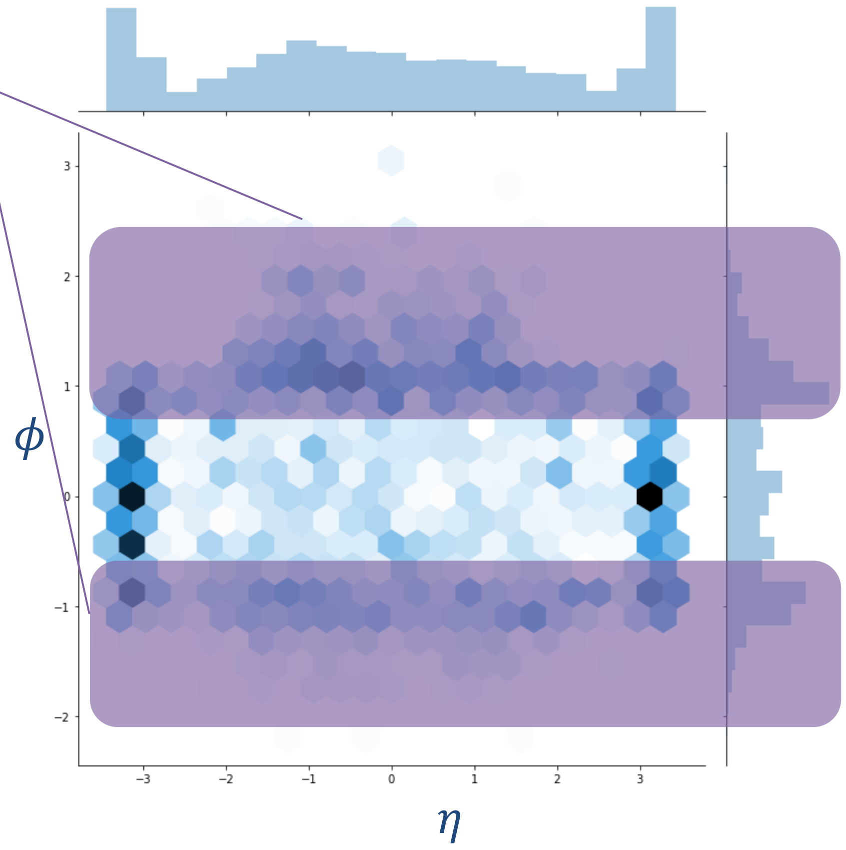
Missing Doublets

All Hits



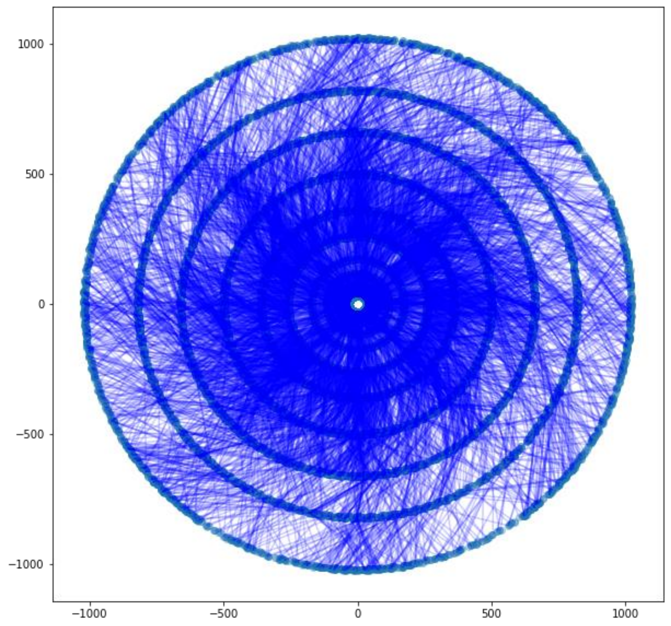
Missing Doublet Hits

Doublets on edge of segments

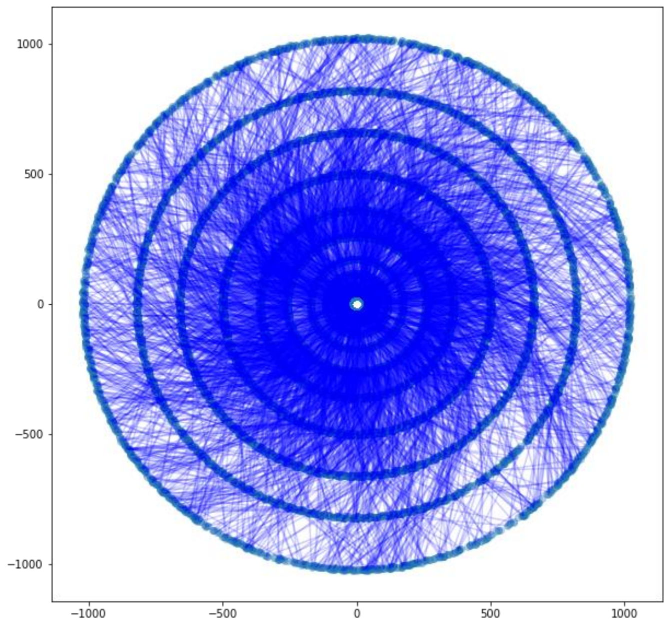


Stitching

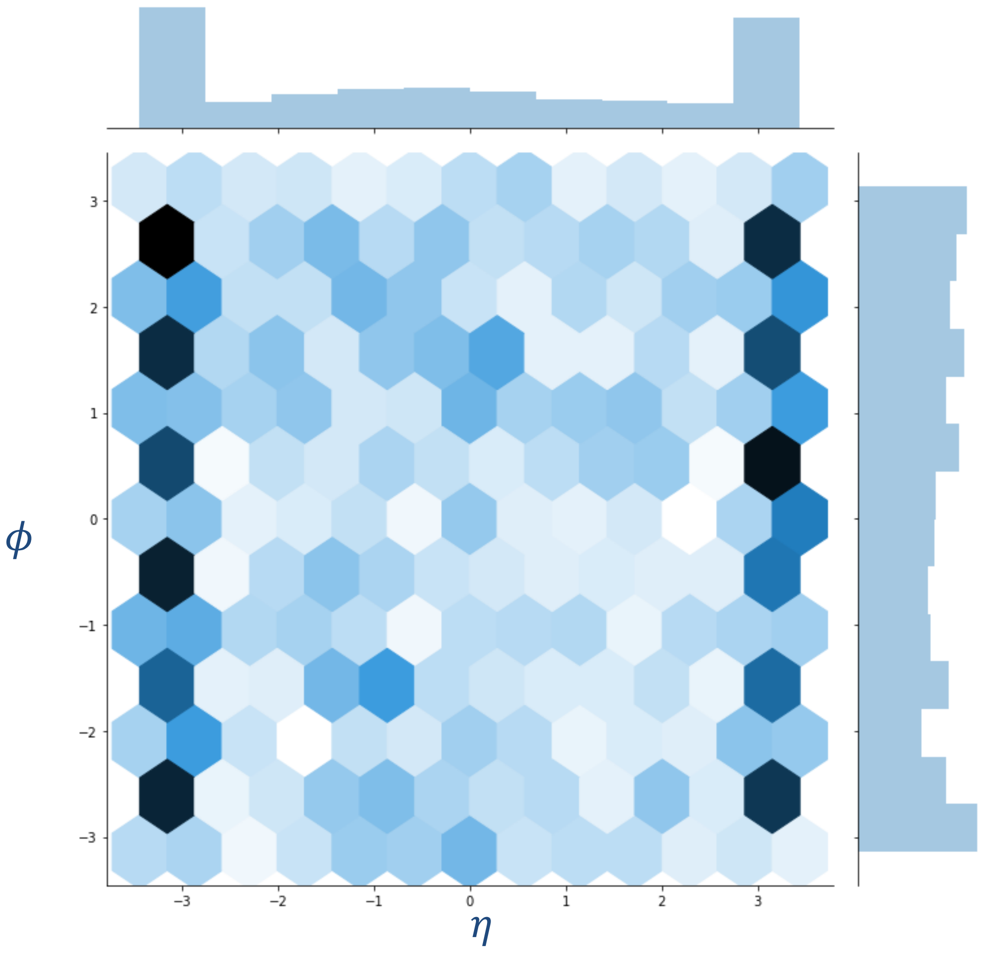
- Significant speed up from eliminated duplicates on edges of segments



Pre-clean-up

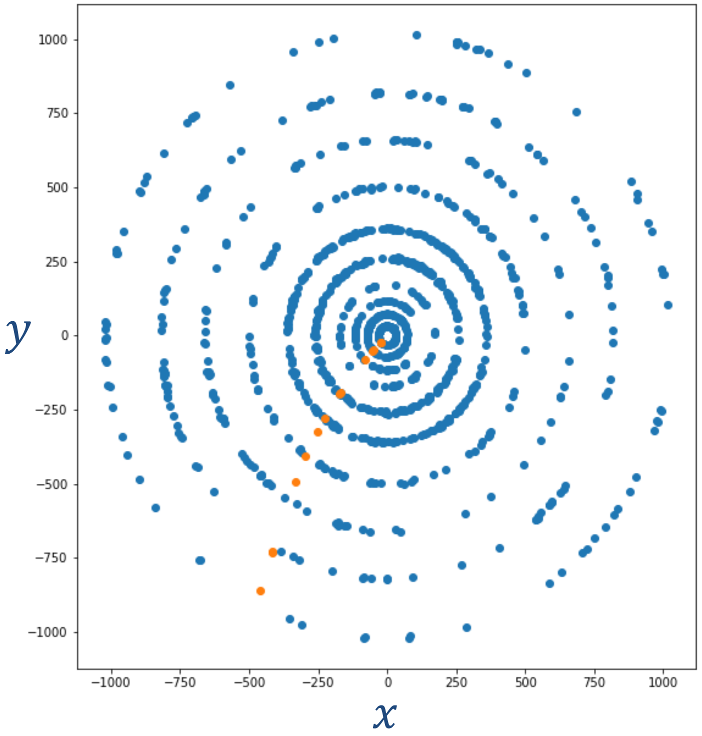


Post-clean-up



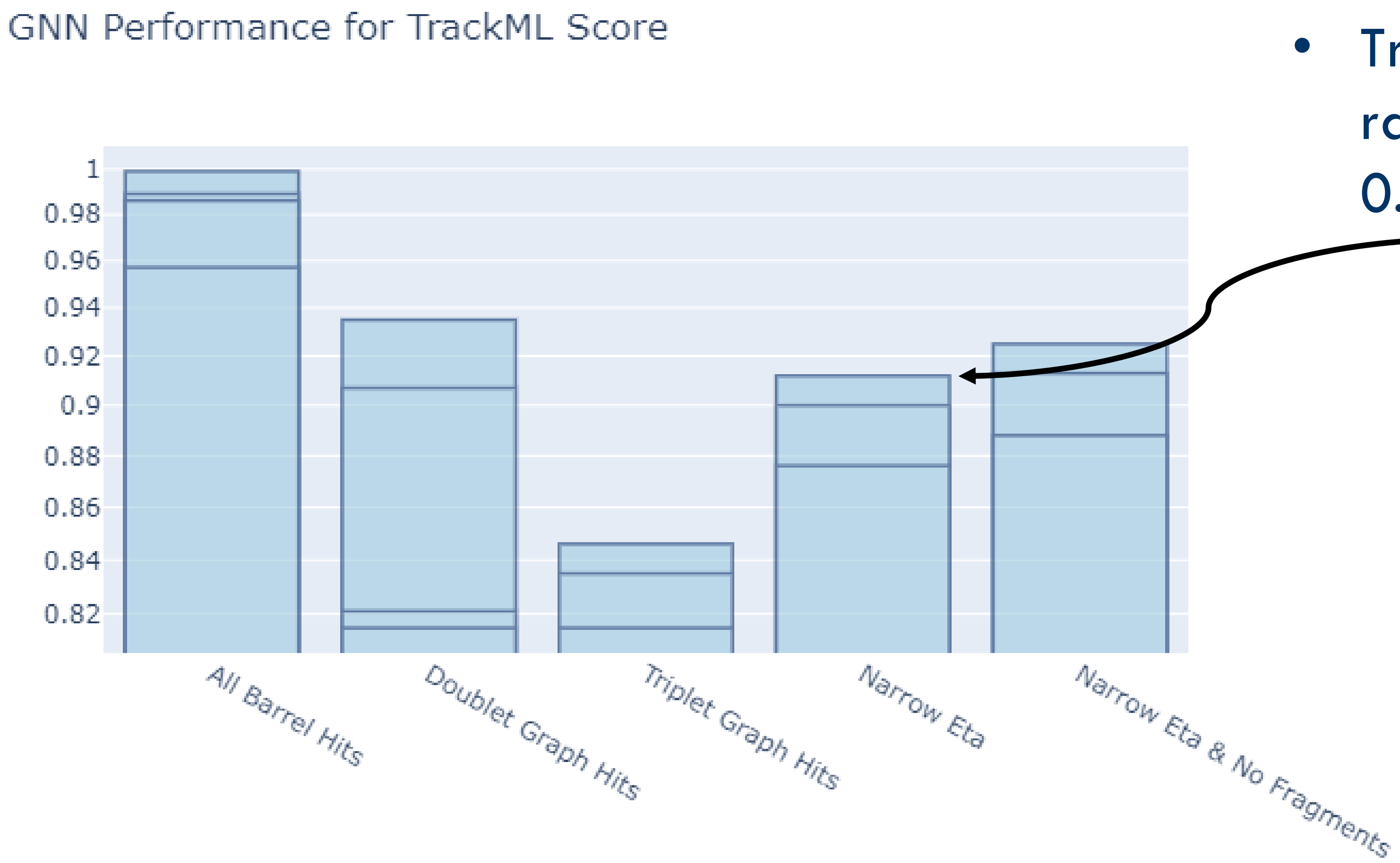
Ignoring Fragmented Tracks

- We throw away all tracks that:
  - Only hit one or two different layers in the barrel
  - Have more than three hits elsewhere in the detector



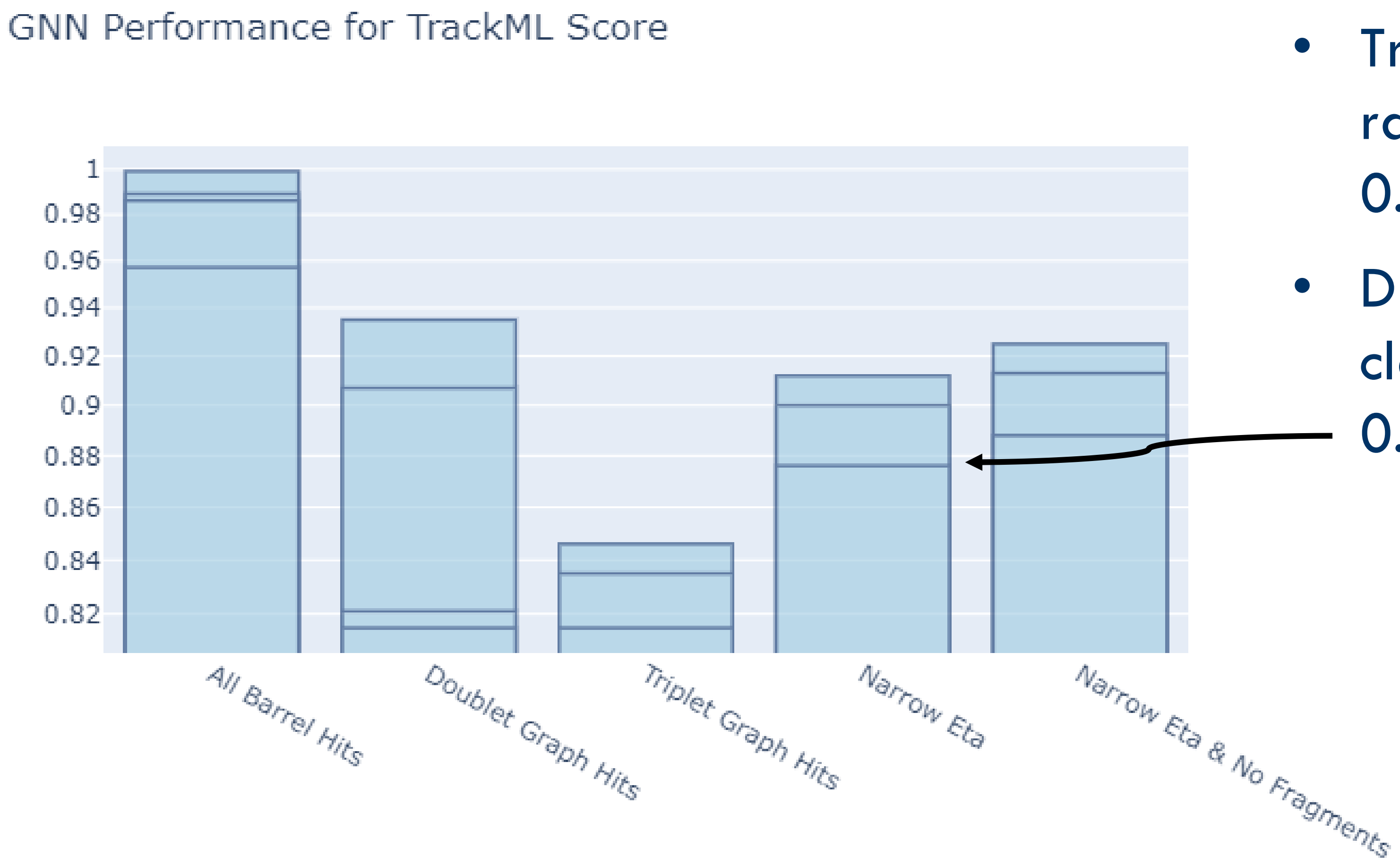
E.g. Although most of this track is outside the barrel, we keep the track to challenge the GNN

Track Labelling: Final-ish Performance



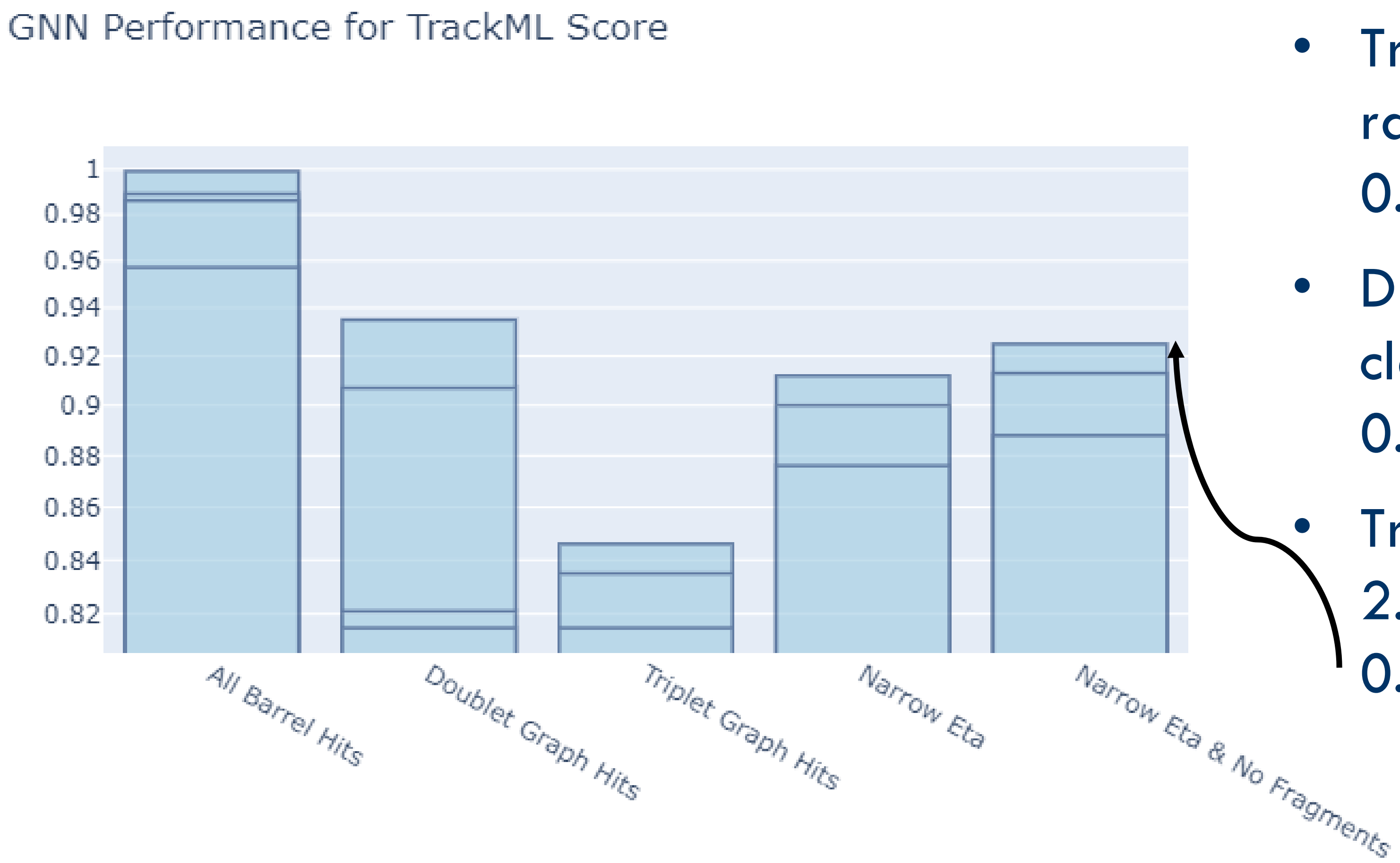
- Triplet graph truth in eta range (-2.1, 2.1)
  0.912

Track Labelling: Final-ish Performance



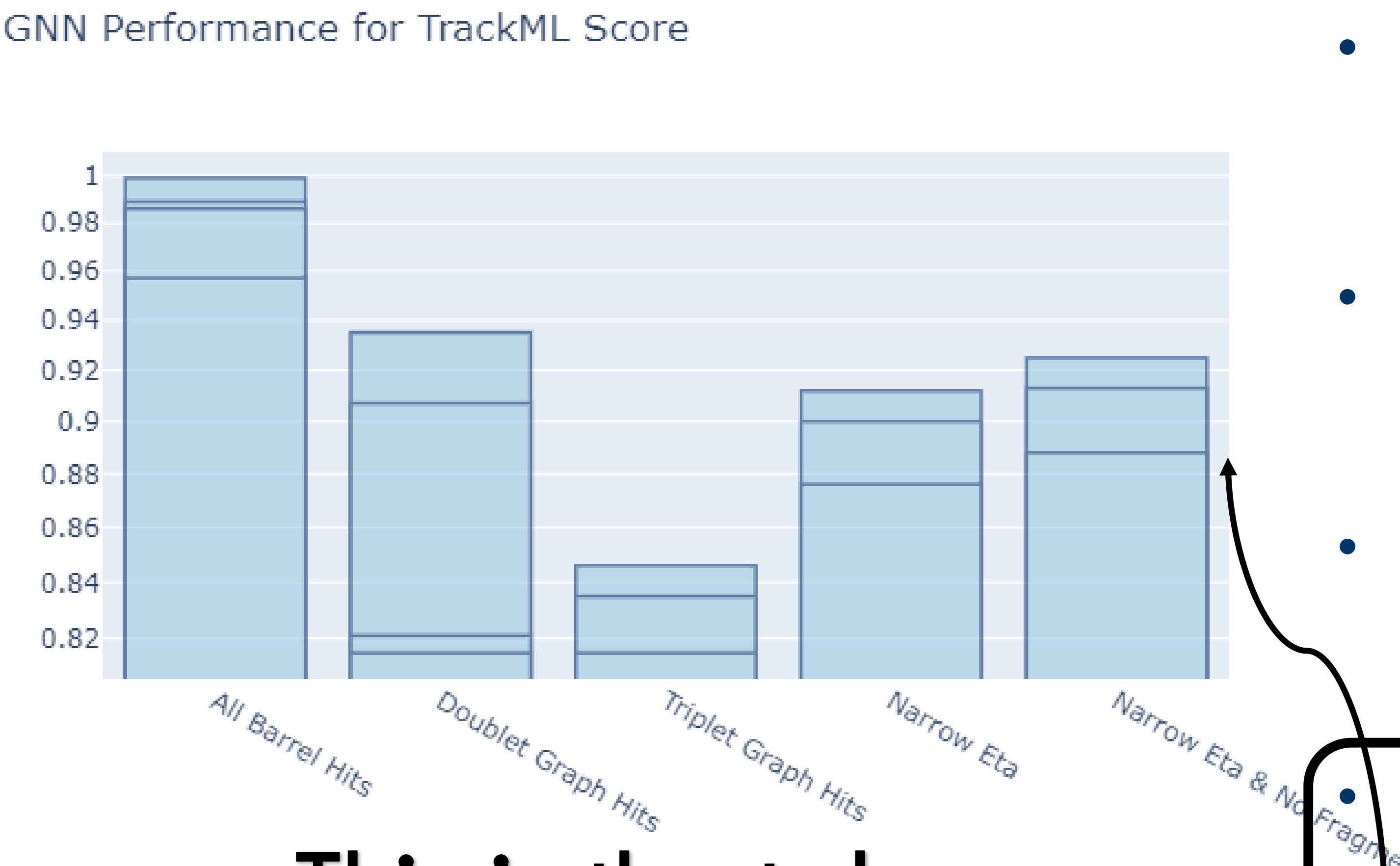
- Triplet graph truth in eta range (-2.1, 2.1)
  0.912
- DBSCAN on triplet GNN classification in eta (-2.1, 2.1)
  0.876

Track Labelling: Final-ish Performance



- Triplet graph truth in eta range (-2.1, 2.1)
  0.912
- DBSCAN on triplet GNN classification in eta (-2.1, 2.1)
  0.876
- Triplet graph truth in eta (-2.1, 2.1) & no fragments
  0.925

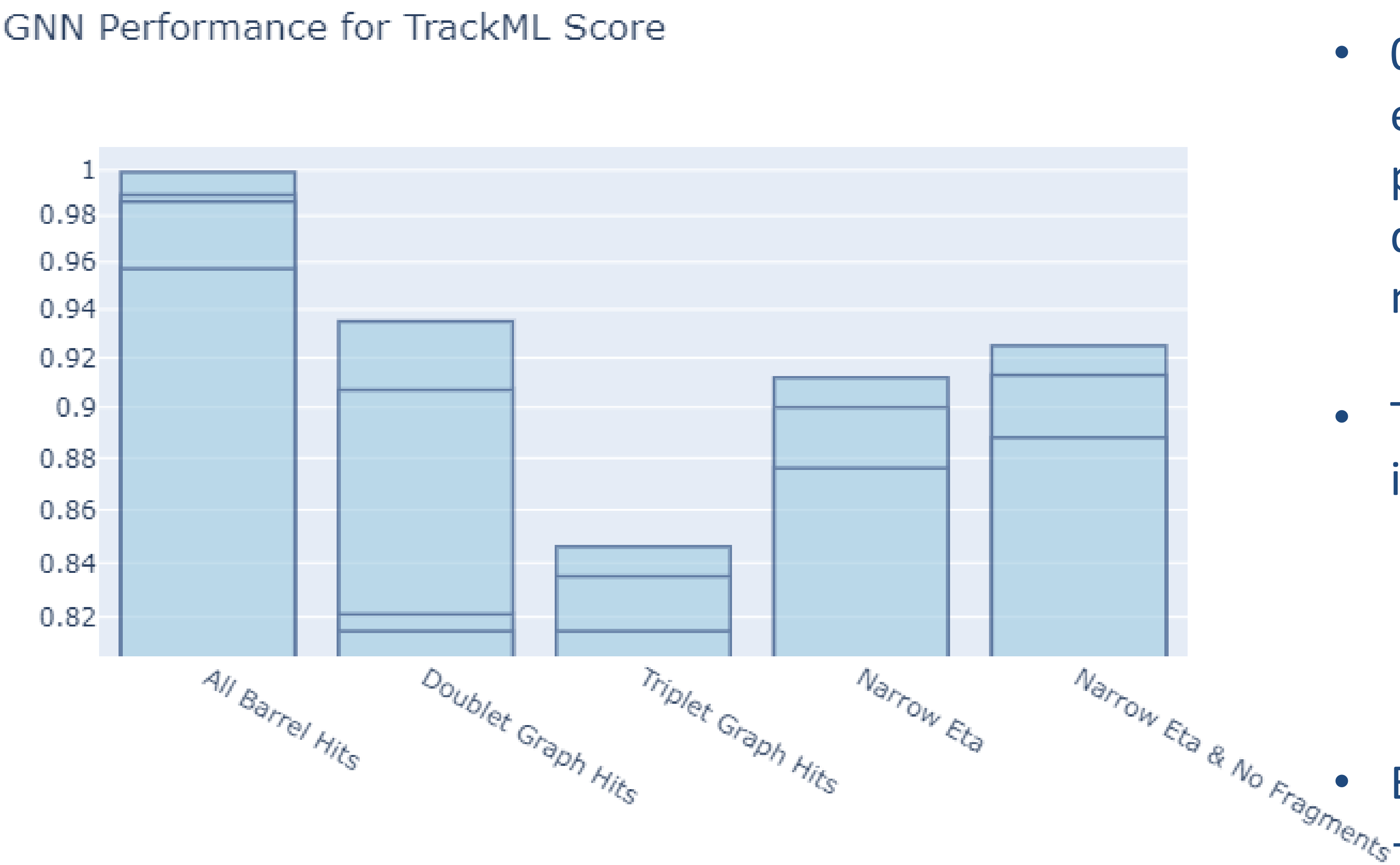
Track Labelling: Final-ish Performance



This is the take-away

- Triplet graph truth in eta range (-2.1, 2.1)
  0.912
- DBSCAN on triplet GNN classification in eta (-2.1, 2.1)
  0.876
- Triplet graph truth in eta (-2.1, 2.1) & no fragments
  0.925
- **DBSCAN on triplet GNN in eta (-2.1, 2.1) & no fragments**
  0.888

Track Labelling: Final-ish Performance



- 0.888 TrackML Score in barrel, emulating whole detector (no punishment for tracks crossing detector volumes) recovers almost all missing doublets

- This is an early result – two big improvement areas are now seen:
    1. Doublet-to-triplet efficiency, and
    2. Embedding construction efficiency

- Every 1% of efficiency gained ≈ + 0.015 TrackML score
- Winning score is 0.922…

Summary

- Seeding pipeline complete, with good performance
- Need concrete comparison with ACTS for CTD
- Track labelling just beginning, with promising performance
- Many low-hanging-fruit optimisations to try and boost efficiency and speed
  - HPO on embedding and GNN
  - Mixed-precision in GNN
  - Include cell features in GNN
  - Some GPU processing with CuPy, but much more could be transferred to work on GPU
  - A multitude of different GNN architectures, one may be especially suited to the physics